Georgia
Institute
of
Technology

RICH ELECTRONIC COMPUTER CENTER / (404) 894-3100 / ATLANTA, GEORGIA 30332

PROGRAMMERS REFERENCE MANUAL
for the
UNIVAC 1108

EXEC 8 EXECUTIVE SYSTEM

February 1972
(Revised)

PROGRAMMERS REFERENCE MANUAL
for the
UNIVAC 1108

EXEC 8 EXECUTIVE SYSTEM

February 1972
(Revised)

PREFACE

This manual is one of a series of manuals prepared by the Rich Electronic Computer Center for the benefit of its users. It is primarily concerned with a description of the Exec 8 Executive System, its control language, and certain programs that act as an interface between the user and the executive to perform utility functions.

Scope

The syntax and semantics of programming languages, such as FORTRAN, COBOL, and ALGOL, are not described here. Additional RECC and UNIVAC manuals provide information in this area.

Application programs such as GPSS, SIMULA, SIMSCRIPT, and Linear Programming are likewise not described. This is partly due to the long and rapidly lengthening list of applications programs available.

Although the basic mechanisms by which language processors (compilers) and application programs are called out are given here, the documentation on each individual component usually gives more specific information.

Since the demand mode of operation, from a Teletype or similar device, cannot be divorced from the executive as a whole, there are many references to demand mode in this manual. However, there are features of the executive pertaining only to demand mode, and these have been avoided. The RECC publication, Demand Terminal Users' Manual for the UNIVAC 1108, presents examples of demand usage and technical information on exclusively demand features. However, the serious demand user will want to reference this manual for detailed information on control statements and system processors used in both batch and demand modes.

Reading Guide

This is a reference manual, not an instructional manual. This means that it is ordered by subject and not by degree of difficulty or utility. Due to the volume of information given, it is not practical to read it from beginning to end.

It is suggested that the novice user begin at the back, with the sample deck setups. These will often supply enough information to begin running programs. Also, they will point out frequently used control statements and FURPUR commands. Through the Table of Contents, the user may find additional information on the constructs that interest or confuse him. Certainly, everyone will read the @RUN, @PWRD, processor call, and @FIN statements.

Sooner or later, most people will tackle the chapter on I/O specification statements. The @ASG statement often confuses new users, but it is the heart of the Exec 8 file system. Don't try to understand the whole chapter the first few times through. Especially avoid F-cycles, because they are rarely used.

Glance through the chapter on System Processors, if only to find out what's available. It will expose you to exclusive Exec 8 features.

You'll undoubtedly be forced to look at the appendix on diagnostic messages. Exec 8 messages are often in a coded form, to help minimize computer overhead.

If possible, sit down at a Teletype and try some things. It's easier to learn when errors can be corrected interactively and when files can be dynamically inspected. Also, the staff of the computer center is available for help.

The first three chapters contain the lowest density of useful information about the system. However, do read about Program Protection and Program Files. Read the rest at your leisure.

In short, don't <u>read</u> the manual, <u>reference</u> it. It would be quite impossible to comprehend the information given without frequently interacting with the system. Here's wishing you an enjoyable and productive relationship with the 1100 series executive system, Exec 8.

Changes in the Second Edition

This is the second edition of this manual. The first edition was published on April 1, 1969, prior to the full-time use of Exec 8 at Georgia Tech. Since that time, much knowledge about and experience with Exec 8 have been gained by both our users and the computer center staff. It is hoped that this revision of the manual will reflect this knowledge and experience.

We have attempted to clarify points in the manual that have confused users in the past. Much of the chapter "I/O Specification Statements" was expanded and rewritten. Answers to repeatedly asked questions, such as "How do I make extra copies of my printout?" and "How do I change the printer margins?" were included. Information regarding the Georgia Tech configuration was added, including descriptions of all mass storage, magnetic tape, and symbiont devices.

The sections on FORTRAN, ALGOL, COBOL, and Applications Programs were removed in the belief that these subjects could be handled better in separate manuals. The remaining sections were renumbered in a more consistent manner than before.

Of course, much of the work involved bringing the manual up to date. The newer features such as batch passwords and saving of catalogued files, and the revised method for saving tapes had to be documented. Also, several enhancements such as @LIST, TD8, and @TSTCAT are now described. Features that were described in the original manual, but are not yet operational, were deleted in hopes of having everything in the manual work as indicated.

We have attempted to create a correct, complete, and useful manual for our user community. Your suggestions on how we may reach higher toward that goal will be greatly appreciated.

Acknowledgment

Most of the material in this manual is reproduced (with appropriate editing for the Georgia Tech environment) with the kind permission of the UNIVAC Division of Sperry Rand Corporation from their EXEC 8 Programmer's Reference Manual, UP 4144, and other UNIVAC publications.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

1. THE EXECUTIVE SYSTEM DESIGN CRITERIA

1.1. Operational Capabilities

To take maximum advantage of the speed and hardware capabilities of the UNIVAC 1108 computer and to make effective use of a given hardware configuration, a complex internal operating environment has been created.

This environment allows for 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 allows for the stringent demands of real-time applications; it is able to store, file, retrieve and protect large blocks of data; and it makes optimum use of all available hardware facilities, while minimizing job turnaround time.

Only through central control of all activities of the UNIVAC 1108 can this environment of the combined hardware and software systems be fully established and maintained to satisfy the requirements of all applications. This responsibility for efficient, flexible, centralized control is borne by the Exec. The Exec controls and coordinates the functions of this complex internal environment and, by presenting a relatively simple interface to the programmer, allows him to use the system easily while relieving him of concern for the internal interaction between his program and other co-existent programs.

1.2. Exec Relation to Other System Components

The UNIVAC 1108 Executive System includes a complete set of source-language processors including FORTRAN V, COBOL, and ALGOL. The operation of all of these processors is controlled by the Exec for the user of the system. By the Executive's assumption of the responsibility for: 1) calling in processors as required, 2) providing inputs to the processors, 3) storage and maintenance of the outputs of the processors, and 4) the integration of activities involving sequences of processor calls, a processor's operation can be confined to the actual processing involved in a single activity. The Executive System will take care of all other functions.

Other components of the 1108 Software System such as SORT/MERGE, APT, PERT/COST, and LP (LINEAR PROGRAMMING) interface with the Executive System in a similar manner.

1.3. Functional Objectives

The primary objectives in the design of the 1108 Executive System are as follows:

(1) To optimize machine facilities usage, and at the same time optimize interaction for all users by the use of multi-programming/multi-processing techniques.

(2) To make available to remote users the complete facilities of the 1108 System.

(3) To provide an Executive Control Language whose structure will allow simple programs to have a simple means of expressing their requirements.

(4) To provide the flexibility to express a complex environment for complex programs.

(5) To provide a broad and easily-used spectrum of program construction, manipulation, and checkout aids, including the permanent storage of program elements on random-access devices.

(6) To provide for tasks to be executed in either batch, demand, or real-time mode.

(7) To provide a simple and flexible means of complete software system generation and maintenance at the individual installation.

(8) To provide system invulnerability to programming error and, as far as is reasonable, hardware errors.

(9) To provide the simplest possible operational characteristics consistent with full utilization of the capabilities.

1.4. Range of Executive System Capabilities

The technical capabilities of the UNIVAC 1108 Executive System span a broad spectrum of data processing activities. Its design is such that no penalties of inefficiency are imposed upon one of these activities by the support provided for the other activities.

1.4.1. Batch Processing

Foremost among system capabilities is the support provided for batch processing. Design emphasis has been placed upon the achievement of ease of run preparation and submission, minimization of job turnaround time, and minimization of operator intervention and decision requirements. Run submission may come from many sources, remote and central. These various

inputs, through the Exec's use and control of efficient multi-programming techniques, may undergo what is essentially simultaneous input, processing, and output. Thus, in a demanding environment, the full capabilities of the 1108 can be utilized efficiently.

1.4.2. Demand Processing

The Exec provides simultaneous use of the 1108 by many users at remote consoles to optimize the user/system interaction rates. Each user shares control of the computational facilities and has the full capability of the 1108 configuration at his disposal.

The demand mode of processing is initiated and controlled by the Executive Control Language. Commands are input via the user's remote console on a conversational basis; that is, an immediate system response will be apparent.

Provisions are made for: (1) Dialed communication connection in addition to leased lines and remote consoles on site; (2) paper tape input allowing pretyped command programs with data for high efficiency communication transmission; (3) user communication with the computer operator and the Exec itself.

1.4.3. Real-Time Processing

A basic responsibility of the Exec is to assist real-time communications (RT/C) programs with Exec functions provided to allow RT/C programs to appropriately influence the Exec and the multi-program background. No attempt is made to generalize the control required in each RT/C program in recognition of the specific tailoring of a RT/C program to both the hardware configuration and the process controlled.

Exec is sensitive to the nature of RT/C processing and provides appropriate mechanisms for: lockout protection from simultaneous record access during program execution; priority sensitivity; protection to RT/C programs from interference because of peripheral access of background programs (search functions, etc.)

Interface with non-standard peripherals can be at the hardware level (I/O commands and interrupts). Exec awareness of individual transmission lines provides for adequate response and flexibility.

1.5. 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 programs may be real-time runs, production runs, classified runs, or simple debugging runs. Infringement of privacy in such a mixture is highly probable especially in cases where debugging runs 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 realtime runs.

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, which normally terminates the program with an IGDM (guard mode violation) message.

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 use of these functions.

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

1.6. Mass Storage Utilization Techniques

The UNIVAC 1108 Executive System is designed to provide installations with an effective and efficient utilization of the mass storage devices available with the 1108. The result is an ability to relieve operators and programmers of responsibilities in maintaining and physically handling cards, magnetic tapes, etc., thus eliminating many of the errors which previously accompanied the use of large-scale software systems. At the same time, the overall efficiency of operation is considerably improved.

Provisions are made for the maintenance of permanent data files and program files on the mass storage devices, with full facilities for modification and manipulation of these files. Security measures are invoked by the Executive System to insure that files are not subjected to unauthorized use. As unused mass storage space approaches exhaustion, provisions are also made within the Executive System for automatic relocation of files of low usage-frequency to magnetic tape. 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. For the most part, dynamic assignment of mass storage space is available to the user via the Executive System. To facilitate efficient utilization of available facilities, the user is also able to return portions of mass storage to general use as he finishes with them.

1.7. Program Files

1.7.1. Basic Concept

The concept of a program file is fundamental to an understanding of the 1108 software system. A program file is essentially a named set of elements. The file name is the prime identifier for the set of elements. To identify and locate the elements within a program file, a Table of Contents is created, and maintained within the program file by the system.

1.7.2. Program File Elements

Within the Table of Contents, each element within the program file is uniquely identified by the following four parameters:

(1) Element type

(2) Element name

(3) Element version

(4) Element cycle

Also included are various parameters such as the date of element creation and the current relative location of the element on mass storage.

The elements contained within a program file are of the following three types:

(1) Source language, or more generally, variable length data images

(2) Relocatable binary

(3) Absolute binary

Typical source-language elements are the following:

(1) FORTRAN source program

(2) COBOL source program

(3) COLLECTOR source element

Any of these elements may be introduced into a program file or manipulated with a file by the use of the appropriate processor (FORTRAN, COBOL, etc.) or by certain utility routines.

The following elements may be thought of as being special-case source-language elements:

(1) COBOL procedure elements

(2) FORTRAN procedure elements

These elements are available to the language processors essentially as source-language library elements. Special elements are required by the system to facilitate the retrieval of source language library elements at compilation time. However, these elements are created and maintained by the system and require no concern on the part of the user.

In addition to the above source elements, sets of executive control-statements may be entered as source elements. These elements may be called by the @START or @ADD statements.

Relocatable elements are the binary output of the processors such as FORTRAN, COBOL, ALGOL, and one special use of the COLLECTOR. Absolute elements are placed in a program file by the COLLECTOR.

1.7.3. Element Name and Version

Each element within a program file is given a name specified by the user. This name is referred to simply as the element name. To distinguish between elements of the same name and type, a user may specify a subname for an element, and this subname is called the element version.

Both an element name and an element version may be from one to twelve characters in length, and these two parameters together must uniquely identify one element among all elements of any particular type. Elements of different types (e.g., source language vs. relocatable binary) may, however, have the same name and version. An element name is required for all elements within a program file (a name is supplied automatically by the Exec in many cases); however, the specification of an element version is not required.

1.7.4. Element Versions

Relocatable elements may be further classified by specifying a class designation which is applied to the version name. The purpose of this classification is the selection of elements based on parameters suited for the particular allocation to be made. Letters within the version names of elements are given meaning by the programmer which can then be used to select a proper class or classes according to the need. Each required element need not be named, but the proper element will be selected by elimination.

1.7.5. "Cycle" Parameter

For differentiation among symbolic elements, an integer parameter called "cycle" is associated with each element. This allows several "copies" of the same version of an element to be retained within a program file. Each item (image) of a symbolic element has a cycle number indicating to which cycle it belongs, and, if deleted, a delete-cycle number to indicate in which cycle this item was deleted. When a symbolic element is updated, the update items are inserted where they belong in the element and given a cycle number one greater than the last cycle of the element. Any previous cycle items that have been deleted by this update are marked so. The user may make references by cycle number. This gives the same effect as though several different copies of the element were maintained. The user may set the number of update cycles to be retained at any level he desires; however, he need set that number if he desires to change it from the standard system assumption. The standard value is five (5).

In specifying a symbolic element for compilation, the user may reference a specific update from a sequence of retained updates by specifying the proper update cycle number as part of the executive control statement calling for the compiler. In compilation, the update entry will be combined with the element in its complete state, thereby creating a complete element as of that cycle.

As soon as the number of updates retained for an element exceeds the specified maximum, the update of the lowest cycle number (the original, complete element) is combined with the update next lowest in cycle number; in effect, the oldest entry is discarded, and the next oldest, in its completed form, becomes the oldest to make room for the latest cycle entry. These corrections thus become incorporated permanently into the basic elements and can only be removed by entering new correction statements.

This technique of handling symbolic elements offers two distinct advantages:

(1) The user is allowed to keep many differing copies of the same element in a program file while requiring little additional storage over that needed for a single copy.

(2) The user is able to refer easily to earlier copies of a specific element without having to prepare corrections deleting previously input corrections. However, if a set of corrections are applied to any cycle except the latest and the updated cycle is to be retained, all cycles that previously followed the cycle to be updated will be deleted. The new cycle number will be the updated cycle number plus one.

2. BASIC CONCEPTS OF THE UNIVAC 1108 EXECUTIVE SYSTEM

2.1. Definitions

Certain terms are referred to in this manual with the assumption that the reader is acquainted with their meaning. The following definitions are for the convenience of the reader.

2.1.1. Activity

A division of a program which may be executed independent of other portions of the program. It is usually considered part of a task.

2.1.2. Activity Registration

The act of registering with the Executive System an activity which can be executed asynchronously with other parts of a program (forking).

2.1.3. Batch Processing

A mode of operation where several runs are grouped prior to processing. Transition from run to run is effected by the Executive System.

2.1.4. Breakpoint

The division of symbiont defined files. Allows those portions of the file to be queued independently of run completion. Maximum use of available printers and punches is achieved in this manner.

2.1.5. Central Site

The 1108 computer and its attached peripheral equipment.

2.1.6. Collection

The process by which elements of a program are collected by satisfying the external symbols of the initial element and all referenced elements. The resulting structure defines a program to be allocated and executed.

2.1.7. Communication Device

An input or output device which operates in a real-time mode. The central processing unit must be prepared to receive input at any time or the information may be lost.

2.1.8. Demand Processing

The manner of processing in which the Executive System or a processor spontaneously reacts to the inputs from a remote inquiry terminal which is sending messages as required. This is essentially a demand and response type of activity.

2.1.9. Element

The basic component of a program file usually defined and manipulated as a unit. The form of an element is dependent upon the program using it.

2.1.10. Executive Control Language

Specifically formatted input information which is used to direct the activity of the Executive System.

2.1.11. Facilities

The peripheral units, main storage, tape drives, drum storage, etc.

2.1.12. File

An organized collection of data stored in such a manner so as to facilitate the retrieval of each individual datum.

2.1.13. Granule

The incremental size in which a storage unit is assignable.

2.1.14. Multi-Programming

The concurrent execution of several programs which occupy main storage. This is accomplished by sharing the attentions of the central processor.

2.1.15. Packet

A contiguous set of words which contain information describing an input/output operation to be performed.

2.1.16. Processor Call Statements

Specifically formatted input information which is used to direct the activity of a system processor. A subset of the Executive Control Language.

2.1.17. Program

A collection of instructions, execution of which results in performance of one or more logical functions. A program is the subdivision of the executable aspects of a run.

2.1.18. Program File

A file in which the data are the constituents of a program or of several programs. This data may consist of program elements in symbolic, relocatable binary, or absolute binary form. Special information in the program file is used to aid the system in the manipulation of the program constituents.

2.1.19. Real-Time Processing

An operating environment in which the response to an external stimulus is sufficiently fast to achieve a desired objective. Depending upon the application, the response time may vary from seconds to microseconds. Generally, real-time processing is under the influence of asynchronous inputs from one or more devices.

2.1.20. Re-entrant Coding

A set of instructions coded in such a manner that they may logically perform the same task on different data sets simultaneously.

2.1.21. Remote Site

A communications terminal which is capable of sending information to and receiving information from the central processor via some common carrier or transmission scheme.

2.1.22. Run

A run is the standard unit in which work is entered into the operating system. This consists of a run command followed by one or more control commands which causes the ordered execution of processors and/or worker programs.

2.1.23. Simulated Fastrand

Drum simulation of Fastrand which allows execution of a program with files designed for Fastrand allocation allocated to the section of the "Flying Head" drum storage designated as simulated Fastrand.

2.1.24. Swapping

The process of storing low priority or suspended programs on secondary storage in order to allow space to retrieve another program into primary storage for execution.

2.1.25. System Processor

A program which performs specialized functions under the control of the Executive System.

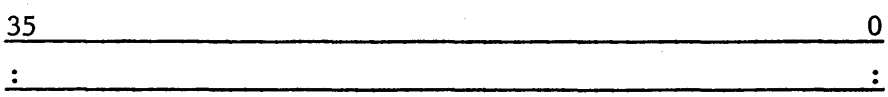
2.1.26. Task

A logical step in the processing of a run. For example, execution of a system processor or a user program.

2.2. System Conventions

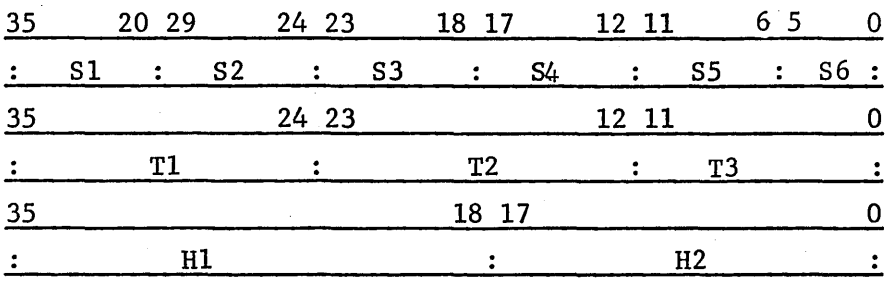
2.2.1. Symbolism

1. When it is necessary to indicate particular bits in a word, they are numbered from right to left,



except in the case of the FORTRAN FLD statement where they are numbered from left to right.

2. When parts of words are referenced the following symbols are used:



3. When referencing an externally defined executive system symbol, the last character is always the $. Procedure names use the $ as their second character. Therefore, it is recommended that the user not use the $ in his symbols.

3. COMPONENTS OF THE EXECUTIVE SYSTEM

The UNIVAC 1108 Executive System is composed of many different routines, each of which performs a specific function. These routines are organized into several separate groups which are the basis of discussion in subsequent sections of this manual. For introductury purposes, a brief description of each component group follows.

3.1. Supervisor

The supervisor controls the sequencing, setup, and execution of all runs. Among those routines included within the supervisor are the scheduling routines, interrupt processing routines, timing routines, and accounting routines.

3.2. Executive Requests

Executive requests are entrances into the Executive System which provide functions for a user program. Depending on the function, it may be performed asynchronously, synchronously, or immediately. If it is not an immediate request, a queue is maintained.

3.3. Symbionts

Symbionts provide the interface between the primary unit record equipment and the user program. These routines are referenced by using executive requests for input and output. Input and output are buffered on the mass storage devices.

3.4. Input-Output Device Handlers

The input-output handlers are responsible for controlling the activities of all I/O channels and peripheral equipment attached to the UNIVAC 1108. These device handlers provide the user with a full capability of peripheral device operations.

3.5. Operator Communications

The communications section of the Executive System handles all communications between the operator and the operating programs. This communication takes place via the computer keyboard and on-line printer on the console channel. Neither the keyboard nor the console printer can be assigned to operating programs.

3.6. File Control System

The file supervisor controls the creation and maintenance of all program and data files. It also maintains an up-to-date master directory of all files catalogued in the system and the availability of all mass storage.

3.7. Data Handling

The data handling routines are designed to process a wide variety of file formats using a general technique.

Files may be processed at the item or block levels with general disregard for the physical characteristics of the I/O device assigned. Data are presented or accepted, randomly or sequentially, on request of the user thereby providing complete operational flexibility for efficient file manipulation.

3.8. File Utility Routines

To aid the user in the manipulation of program and data files, a set of file utility routines is provided by the Executive System. These routines perform a variety of functions for system and user data file maintenance.

3.9. Auxiliary Processors

A set of auxiliary processors is included in the executive system. These processors complement the source language processors such as FORTRAN. This set of processors includes the COLLECTOR for linking relocatable sub-programs, and the PROCEDURE DEFINITION PROCESSOR for inserting and modifying COBOL or FORTRAN procedure definitions in a program-file.

3.10. Processor Interface Routines

The processor interface routines provide a simple, standard interface for all processors within the system. Complete facilities are provided for the input of source-language statements and the output of the resulting relocatable binary code.

3.11. The Diagnostic System

A comprehensive diagnostic system is available within the 1108 Executive System to aid the checkout of user programs. Commands are available which can trigger snapshot dumps at the time of compilation or collection of a user routine. Post-mortem dumps are also available through an Executive Control Statement.

4. EXECUTIVE CONTROL LANGUAGE

4.1. Purpose

Control of the operating environment on the UNIVAC 1108 is accomplished through a set of control statements. These statements direct the executive in scheduling, assignment of facilities, and in the disposition of program and data files. The language is designed in a compact and descriptive manner to facilitate use and yet provide all of the features and functions of a modern Executive System.

4.2. Statements

4.2.1. General Content

The basic format of the Executive Control Statements is quite simple and is amenable to a large number of input devices. Statements are not restricted to a card-image format; hence, they may be of variable lengths. Each statement consists of a recognition character in column one, followed by a command which categorizes the statement, followed by a variable number of specifications fields, and concluded by a comments field. The recognition character is a master space (@), which is a multiple (7-8) card punch or its equivalent for other types of input devices. The end of a statement is signified by the end of a card for card-image input, or by a carriage return or its equivalent for other types of input devices.

Executive Control Statements are always logged in a batch run's print file. If a control statement is in error, the diagnostic is printed immediately following the statement.

4.2.2. Statement Format

The general format of an Executive Control Statement is:

```
@LABEL:COMMAND, OPTIONS   SPEC1, SPEC2,...,SPECN   COMMENT
```

The following gives a description of each of the "fields" of the Executive Control Statement as well as format and continuation rules.

4.2.2.1. 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 alphanumeric set (A...Z, 0...9), the first of which must be an alphabetic. If a label is specified, it must be immediately followed by the colon (:). A label is used only when dynamic adjustment of the control stream is required. The discussion of their use is deferred to the section entitled 'Conditional Statements'.

4.2.2.2. Command Field

The command field must always be specified as it determines the statement's basic operation. The command is limited to six characters from the alphanumeric set (A...Z, 0...9), the first of which must be alphabetic. For certain control statements, the options field, which is an appendage to the command field, is recognized. When the options field is specified, the command field terminator is the comma (,). However, if an options field is not specified, blank ( ) is the command field terminator.

4.2.2.3. Options Field

The options field provides the user with the ability to specify certain options, in the form of unsequenced alphabetic characters, to the particular processor addressed in the command field or to a specific program as it is executed. On some control statements the options field 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 specifications fields.

4.2.2.4. Specifications Fields

The specifications fields of an Executive Control Statement are separated by commas and are specified by the user as dictated by his requirements. The content of each specification field, the number of specification fields, and whether each is required or optional, varies with the command selected. Specification fields, in turn, may contain 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.

In many cases, the specifications on a control statement will be a file name or element name. In the remainder of this manual, the following conventions apply (where brackets enclose optional fields):

FILENAME is used to indicate

[[QUALIFIER]*]FILE[(F-CYCLE)][/[READ-KEY][/WRITE-KEY]]

ELTNAME is used to indicate

[FILENAME.]ELEMENT[/VERSION][(ELEMENT-CYCLE)]

Qualifier, file, element and version names are 1-12 alphanumeric characters ('$' and '-' are also allowed). Keys have 1-6 characters from the entire Fieldata character set, excluding only space, comma, slash, period, and semicolon. F-cycles are numbered upward from 1; element cycles are numbered upward from 0.

When the qualifier is omitted, the USER-NAME from the @RUN control statement is used, except in the special case where a leading asterisk appears before the filename and a qualifier has been previously furnished on a @QUAL statement. When the F-cycle or element-cycle is omitted, the most recently created cycle is used.

When the filename portion of an eltname is omitted, the system usually assumes an implicit reference to the run's temporary program file, TPF$. TPF$ is automatically assigned to every run by the system.

Although the distinction between filenames and element names is often evident from the context, there are many cases where a period must follow a filename, or it will not be accepted, or wrongly treated as an element name. In such cases, 'FILENAME.-or-ELTNAME' will be used. A period may always follow a filename, except on a @BRKPT statement.

4.2.2.5. Leading Blanks

Leading blanks are allowable following the recognition character (@), the colon (:) if a label is specified, the field separator (,), and the subfield separator (/). A blank in any other position acts as the separator signifying the start of the specification fields or comments field. An empty field or subfield is one that contains no characters or one or more blank characters. When all remaining fields or subfields are empty, they may be omitted.

4.2.2.6. Comments Field

At least one blank character must precede the comment field. The comment itself may contain any character except the semicolon (;), the continuation character. The comment field is ended by end-of-card or its equivalent for other input devices. The comment field is never required. If specifications fields are omitted, the comment field must begin with a period (.) followed by a blank. This is also true when the content of a specifications field is unrestricted and variable in length (as with the

@MSG statement). The @XQT statement is an example of a statement where specifications are possible but may be omitted.

Note: The above paragraph has described the comment field separator as follows:

@ control card ¢.¢ comments

4.2.3. Continuation Rules

In certain situations, a statement may require more than one line or card. In such cases, coding of a semicolon (;) indicates continuation on the next card or line. A statement may be split at any point, after the options field, where a leading space 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 (@) should not be placed in column one on a continuation line.

4.3. Statement Types

4.3.1. General

The 1108 Executive System recognizes five types of control statements:

(1) Organizational statements;

(2) input/output specifications statements;

(3) processor call statements;

(4) program execution statements; and

(5) conditional statements.

Each statement is discussed individually in succeeding paragraphs. The order of presentation is as shown in the table below.

SUMMARY OF EXECUTIVE CONTROL STATEMENTS

Statement Type	Command	Usage
Organizational Statements	* @RUN	Appears at the beginning of each run. Provides accounting, scheduling, and ID information.
	@FIN	Appears at the end of each run.
	@MSG	Places a message on the central-site console typewriter.
	@HDG	Used to place a heading line on print output.
	@ADD	Used to dynamically expand the run stream.
	@START	Used to schedule the execution of an independent run.
	@SYM	Used to schedule non-standard symbiont action.
	* ** @COL	Used to specify form of input, e.g., key-punch code.
	* ** @PWRD	Used to specify the user's batch password.
	* ** @ BIN	Used to specify the bin number for returning output.
	@LOG	Used to enter text into the system log.
Input/Output Specification Statements	@ASG	Used to assign a particular input/output device or mass storage file to a run. There are two types of @ASG statements; mass storage and tape. Also used to catalogue Mass Storage files.
	@MODE	Used to change the mode settings (density, parity, etc.) of a tape file.
	@CAT	Catalogues Fastrand files.
	@FREE	Used to deassign a file and its input/output device or mass storage area.
	@USE	Used to set up a correspondence between internal and external file names.
	@QUAL	Used to define a standard file name qualifier.

* These control statements cannot have labels. The asterisk is not part of the statement.

** These cards are fixed format. The asterisks are not part of the statement.

SUMMARY OF EXECUTIVE CONTROL STATEMENTS (Continued)

Statement Type	Command	Usage
Processor Call Statements	@PROCESSOR NAME	Used to execute a processor. @COB for COBOL compiler, @FOR for FORTRAN, @ALG for ALGOL, etc.
	@MAP	Used to call the COLLECTOR and prepare an absolute element.
	@XQT	Used to initiate the execution of a program.
	* @EOF	Used to separate data within the control stream.
	@PMD	Used to take edited post-mortem dumps of the program just executed.
Conditional Statements	@LABEL:	Used to attach a label to an existing control statement.
	@SETC	Places a value in the 'condition' word.
	@JUMP	Used to branch control within the control stream.
	@TEST	Used to test the 'condition' word in the course of deciding the effective control stream.

* These control statements cannot have labels. The asterisk is not part of the statement.

4.4. Organizational Statements

4.4.1. The @RUN Statement

The @RUN statement must be the first statement of each run. Its purpose is to identify the run and to furnish parameters necessary for scheduling and accounting purposes. The format of the @RUN statement is:

```
@RUN,PRIORITY/OPTIONS RUNID,REFERENCE-NUMBER,USER-NAME,RUN-TIME,PAGES/CARDS,START-TIME
```

On the @RUN statement the normal options field is divided into two subfields separated by a slash (/). The first subfield specifies the 'PRIORITY' of the run and the second specifies the 'RUN-OPTIONS.'

The 'RUNID', 'REFERENCE-NUMBER', and USER-NAME' fields are the only specification fields that need to be specified by the user. For demand runs, the 'START-TIME' field is not honored.

4.4.1.1. PRIORITY Subfield

The priority subfield contains an alphabetic character. At system load time, the following information is specified for each 'REFERENCE-NUMBER' and 'USER-NAME': (1) the highest priority letter allowed for this 'REFERENCE-NUMBER' and 'USER-NAME' (2) priority to use if none is specified on the @RUN statement.

The highest priority allowed and the assumed priority are the same for each 'REFERENCE-NUMBER' and 'USER-NAME'. If the priority on the @RUN statement is higher than allowed, the run is terminated immediately. If the priority subfield is left blank, the priority character is chosen as specified in the 'REFERENCE-NUMBER' and 'USER-NAME' file entry.

4.4.1.2. OPTIONS Subfield

The run 'OPTIONS' subfield may be used to place certain constraints on the run. This field is never required and when left blank, normal system action occurs. The available options are:

- N - disallow all postmortem dumps. Specification of this option for runs containing no @PMD statements saves the overhead of dumping core to the diagnostic file at termination of every user program.

- Y - allow postmortem dumps of system processors.

If neither N nor Y options are specified, @PMD's will be allowed of all programs except system processors (FOR, ALG, COB, MAP, FURPUR, etc.).

X - Do not automatically reschedule run if it is active during a system failure. This option should be specified if rerunning a partially completed job might destroy data. Appropriate @MSG statements to the operator should also be included in the deck.

S - Sequenced runs. If two or more runs must be run in sequence (i.e., a run must finish before the next run can be started), the S option should be used on all run statements except the first. A null bin card must separate the run decks. See Sample Deck Setups for an example of sequenced runs.

4.4.1.3. RUNID Field

The 'RUNID' (identification) field must be specified to identify the run to the system. This field is limited to a maximum of six characters from the alphanumeric set (A...Z, 0...9).

If the system finds that a run being submitted has the same runid as a previous run that has not finished execution, the executive will assign a unique runid to the run, notify the operator (and the user, if demand) of the change, and continue processing the run. The new ID is used for all operator-executive communications concerning the run. Normally, the new ID is established by adding an alphabetic character if the original ID is less than six characters. If the original ID is six characters, the rightmost character is replaced.

If a batch @RUN statement is preceded by a standard @ BIN card, the bin number (possibly modified for uniqueness) is used for the runid. This insures that all print files created by this run, and by runs @STARTed by this run, will bear the proper bin number.

The demand user should use his department abbreviation as the first two characters of his runid to ensure that any output printed onsite as a result of the demand run will be returned to the department's permanent bin. The department abbreviation is the first two characters of the password; hence the user may determine his department abbreviation from his password.

For a batch run, the @RUN statement must be followed immediately with a @PWRD statement.

For a demand run, run initiation is accomplished as follows:

1. Dial the 1108 -- receive "beep" tone.
2. Enter siteid assigned to terminal -- receive "1108" message.
3. Enter password -- receive "NO RUN ACTIVE".
4. Enter a @RUN statement.

4.4.1.4. REFERENCE-NUMBER Field

The reference number field is used to specify accounting codes, and it must be filled. The field contains from one to twelve characters from the set A...Z, 0...9. The user must use the NON-BLANK characters assigned by RECC.

4.4.1.5. USER-NAME Field

The user-name field classifies the run for accounting purposes and permits insertion of the implied qualification of file names when no specific qualification is given. This field is limited to 12 characters from the set A...Z, 0...9, and -. This field must be specified. The user must use the non-blank characters of his name as submitted by his School or Department and approved by RECC. For a more detailed explanation of the use of this field as a file qualifier, see the chapter on INPUT/OUTPUT SPECIFICATION STATEMENTS.

A set of allowable names and reference numbers is created at system load time. A run is accepted if its name and reference number are known to the system. If not, the operator is notified and the run is rejected.

4.4.1.6. RUN-TIME Field

Use of the run-time field is optional. The run-time field specifies the programmer estimated number of minutes of central processor unit (CPU) time required for the run. If this time is exceeded, as measured by the time that the run has control of the CPU, the run is terminated immediately. The user must not request more time than is specificied for his particular Section on RECC form 1 or 2 (see your Departmental Computer Coordinator). If this field is omitted, an estimated RUN-TIME of one (1) minute will be automatically supplied. If the user wishes to specify his run-time in seconds instead of minutes, he may do so by prefixing his number with the letter S; e.g., S60 means 60 seconds or 1 minute.

4.4.1.7. PAGES Subfield

Use of the pages subfield is optional; it provides the system with a page number estimate of printed output that the programmer is expecting. If this subfield is omitted, a maximum of 50 pages will be assumed. The run will be automatically terminated when the estimate is exceeded. The user must not request more pages than is specified for his particular Section on the RECC form 1 or 2. (See your Departmental Coordinator.)

4.4.1.8. CARDS Subfield

The use of the cards subfield is identical to the pages subfield except that it applies to the number of punched cards expected during the run, rather than the number of printed pages. If this field is omitted, 100 cards is assumed for all jobs. Note that CARDS is separated from PAGES with a slash, not a comma. If a comma is used, the specification is interpreted as START-TIME.

4.4.1.9. START-TIME Field

The start-time field is used to specify (delay) the time at which the run will be considered for execution. In the absence of a start-time specification, which is the normal case, the run is considered for execution immediately. When a start-time is specified, the run is not included in those available for execution until the start-time has arrived. At that time, it is considered for execution according to the given priority.

The start-time is based on a 24 hour clock. If a 'D' precedes the time specification, it is taken as the time of day; otherwise, it is taken as the elapsed time from run submission. The time is given in hours and minutes and cannot exceed 2400 (24 hours, 0 minutes). For example, a specification of 'D910' would be taken as 9:10 a.m., and 'D2110' would mean 9:10 p.m.

The start-time field allows a run to be submitted with the assurance that it will not be executed prior to the given time. This feature is desirable when input data are not yet ready but will be by start-time.

4.4.1.10. RUN Restrictions

The following is a summary of executive action concerning run restrictions: A run will not be processed if it contains an invalid 'USER-NAME', 'REFERENCE-NUMBER', or password, or an invalid priority, time, or page request. All runs will assume a 100 card punch estimate if the 'CARDS' subfield is not specified. All runs will be automatically terminated when the estimated time, pages, or cards is exceeded.

4.4.1.11. @RUN Statement Examples

Consider the following @RUN statement examples:

(1) @RUN R231, 51C12009, DOE-D-J, 10, 100

The options field is not used, meaning that the priority will be the assumed priority for this name and reference number and the run-option is not required. This is run R231 for name 'DOE-D-J' and reference number '51C12009'. The estimated running time is ten (10) minutes and the estimate of print output is 100 pages.

(2) @RUN,X WCMI,99A650,SMITH-L

The priority code is 'X', the runid 'WCMI' for name 'SMITH-L' and reference number '99A650'. The time and page estimates of one minute and 50 pages respectively will be used.

(3) @RUN KPM2,57CI12001,JONES-R-T,,/300,D1300

Run 'KPM2' of name 'JONES-R-T' and reference number '57C12001' is to be processed. The priority is to be the maximum allowed for this name and reference number. Time and page limits of 1 minute and 50 pages will be assumed. This run will punch a maximum of 300 cards. The run will not be considered for execution until 1:00 p.m.

(4) @RUN,T/S TSTACT,89C12005,ALBERT-F-S,15,300/50

Run 'TSTACT' of name 'ALBERT-F-S' and reference number '89C12005' is to be processed and has a priority of 'T'. The estimated running time is 15 minutes, estimated print output is 300 pages, and estimated punch output is 50 cards. It will not be considered for execution until the previous deck has been processed provided that the @RUN card was preceded by a null bin card. Otherwise the job will be processed immediately.

(5) @RUN EC8LT,61C12049,SMITH-G,,,30

Run 'EC8LT' of name 'SMITH-G' and reference number '61C12049' will not be considered for execution until 30 minutes after the deck has been read in by the computer. The priority will be the maximum allowed for this name and reference number. The time and page limits will be 1 minute and 50 pages respectively.

4.4.2. The @FIN Statement

The @FIN statement is used to signal that the end-of-run has been reached. It is required with all runs and must appear as the last statement. It is never passed as a data image for @ELT or @DATA. This statement cannot be continued on a second card or line.

The @FIN statement's format is:

```
@FIN
```

When the @FIN statement is encountered by the coarse scheduler, the accounting routines are entered and all remaining facilities, temporary files, and core space are released. Note: For onsite batch, this card must be the red @FIN card provided by the computer center.

For a demand run, the @FIN statement will terminate the current run and wait for a new run to be started (by entering a password and @RUN statement). If no new run is to be started, the user should hold down the CTRL key and press the EOT ("D") key. The user may alternatively press CTRL/EOT without entering @FIN, and the executive will automatically assume a @FIN.

4.4.3. The @MSG Statement

The @MSG Control Statement is used to type a message on the central site console typewriter. It has the form:

```
@MSG,OPTIONS                    MESSAGE
```

The message has a maximum length of 50 characters. The first non-blank character is the beginning-of-information and the end-of-information is the last character prior to the end-of-line, the comment field or the 50 character maximum whichever occurs first. The @MSG statement can be used to direct the operator in such areas as disposal of output, abnormal or undocumented procedures, etc. The message is prefaced by the runid when typed.

The @MSG statement may contain the following options:

- W - Causes the run to be held until the operator responds to the message. The operator may respond with up to 50 characters. His response is printed immediately. If the operator cannot comply, he may abort a batch run via the keyin 'X'.

- N - Causes suppression of the typing of the message on the console typewriter. In this case the statement is listed on the printer only. When the N option is present, the W option is not effective.

The W option can be used to direct the operator in the loading and general management of peripheral devices (in those cases not automatically taken care of by the executive), and in communicating directly with the operator from a demand terminal.

The N option can be used to simply place a message on the printer or as a way to suppress console action without removing the @MSG statement.

An example of the @MSG Control Statement is

```
@MSG EXPECT 2 REELS OF OUTPUT FOR FILE XYZ
```

Another example, where the operator must respond, could be:

```
@MSG,W  IS REMOTE HOOKUP READY
```

4.4.4. The @HDG Statement

This control statement provides the user with an automatic means of printing a heading on each succeeding page of the print file. The format of this statement is:

```
@HDG,OPTIONS      HEADING-TEXT
```

The allowable options are:

- N - Turn off printing of the heading.
- P - Begin page number with 'page 1'.
- X - Do not print date or page count.

The 'HEADING-TEXT' field is variable in length with a maximum of 96 characters allowed. This field is separated from the control field with a single space, thereby allowing leading spaces in the text. The end of the text is denoted by the last character prior to the end-of-line, or the comment field, or the 96 character maximum, whichever occurs first.

The heading is printed on the second line above logical print line 1. If this upper margin is one line or non-existent, the heading will not be printed. The date and page number will appear to the right of the heading text. A page count for each print file is maintained by the processing symbiont. When heading is specified without the 'P' option, the page count current to the file is used to begin page numbering. Any number of @HDG statements may appear in the control stream.

A period in the HEADING-TEXT field signifies the end of the printed heading and the beginning of the print-control subfield.

4.4.4.1. Print-Control Functions

Print-control functions may be specified on a @HDG statement, formatted as follows:

```
@HDG,OPTIONS      HEADING-TEXT.PRINT-CONTROL
```

Operation of the @HDG options and text is as described above. The 'PRINT-CONTROL' subfield is a string of functions, each of which begins with a function letter, has its parameters separated by commas, and ends with a period.

The available functions are:

```
L, line-number-to-which-printer-should-be-spaced.
M, number-of-print-lines, top-margin, bottom-margin.
S, text-requesting-special forms.
```

'L,1.' in the 'PRINT-CONTROL' subfield will cause a pageup after the @HDG statement is printed. 'M,66,6,3.' will reset the print margins to the standard if they have been changed by a previous print control function.

4.4.5. The @ADD Statement

The @ADD Control Statement provides a means of inserting images into the control stream from any file or element in the System Data Format. These files may contain data or any control cards allowed in a run stream. The file being added may have been created by the @DATA statement, the @ELT statement, or a user program. The images in the file being added need not exist until the @ADD command is executed. This means that the user is free to have worker programs in the first part of a run generate files to be added later in the run.

The format of the @ADD Control Statement is

```
@ADD,OPTIONS        FILENAME.-or-ELTNAME
```

where 'FILENAME' may be the name of the file if an entire file is to be added, or it may be replaced by the standard reference to an element '[PROGRAM FILE.] ELEMENT[/VERSION][(CYCLE)]'.

The allowable options are:

- D - Used in data mode (@DATA or @ELT,D in control) to cause the specified file or element to be added. If the D option is not used in data mode, the @ADD statement itself will be used as a single input image.
- E - Used primarily when adding images to the run stream which are to be read by an executing user program. After the last image in the added data has been read, a subsequent read request returns an end-of-file status, as if there had been an @EOF control statement at the end of the added data.
- P - Print the @ADD statement. The @ADD statement is always printed if an error is detected.

When the @ADD Control Statement is encountered in a control stream, the first image of the added file replaces the @ADD control image. All subsequent control stream images will be taken from the added file until the end of file or, if an element is being added, until the end of the element is encountered. Following the end of the added file, the control stream is automatically resumed at the image following the @ADD statement.

@ADD statements may be nested 3 deep provided there is no attempt to add a given file (or element) twice in the same nest. When this occurs, or when a non-existent file is specified, the run is terminated, if batch.

The @ADD feature is of particular value to the remote user (batch or demand) in that control statements and/or data can be submitted only once but used in many subsequent runs.

The following list of control statements are considered illegal within an @ADD file: @RUN, @COL, @FIN, @ BIN, @PWRD.

4.4.6. The @START Statement

The @START statement affords the user a means of scheduling of one or more runs from within a run control stream. Runs to be scheduled in this manner must be catalogued data files created by the @DATA processor or a user program or they may be elements of a catalogued program file created by the @ELT,D statement. The run file and the run element are in system data file (SDF) form.

The @START feature can be used when one run must generate a data file for input by another. In fact, the generating run may elect to build a catalogued file containing an entire run control stream and then call for it to be scheduled. Notice that the @START statement can be used to allow the parallel processing of certain operations, since tasks from different runs can be executed concurrently.

It may also be employed by demand terminals as a means of initiating a batch run whose control stream has been previously entered into the system as a data file, thus eliminating the necessity of retyping the required control statements. The @START is of particular benefit in initiating prestored utility routines and standard production runs.

In its simplest form, the @START statement's format is:

```
@START     NAME, SET
```

where NAME is FILENAME.-or-ELTNAME.

The 'NAME' field must be either a data file name or an element name in the standard format for symbolic element description. The 'SET' field can contain an octal number to be 'SET' in the condition word of the run being scheduled in order to determine the effective control stream (see section on Conditional Statements). The 'SET' specification is never required. The referenced stream must begin with a @RUN statement for this new, independent, asynchronous run. A @PWRD statement must <u>not</u> be included in the referenced stream. The end of the file or element denotes an implied @FIN. When scheduling such a run, it is sometimes desirable to be able to change some of the parameters on the @RUN statement that heads a prestored control stream. The user may want to supply parameters such as the time and page estimates.

A substitution can be made for certain parts of a prestored @RUN statement by the use of a more complex @START statement of the form:

@START,PRIORITY/OPTIONS NAME,SET,,,,RUN-TIME,PAGES/CARDS,START-TIME

Note that the statement has the same format as the @RUN statement except that the file 'NAME' field and the 'SET' field precede the 'RUNID' field. Another notable difference from the @RUN statement is that all fields are optional except the file 'NAME' field. All non-blank fields will be substituted in place of those on the prestored @RUN statement.

A substitution is always made to replace the 'RUNID', 'USER-NAME', and 'REFERENCE-NUMBER' on the prestored @RUN statement with the respective fields taken from the @RUN statement of the initiating run.

The replacement of the 'RUNID' is done to insure that output created by the @START run is returned to the proper bin. The replacement of the 'REFERENCE-NUMBER' and 'USER-NAME' fields is done to insure that unauthorized use of a reference-number/user-name combination is not made.

4.4.7. The @BRKPT Statement

The @BRKPT statement is used to close out one portion of the PRINT$ or PUNCH$ file and start a new part. If the part of the file being closed is system defined, it will be automatically queued for the proper symbiont.

The formats of the @BRKPT statement are:

(a) @BRKPT PRINT$-or-PUNCH$[/FILE]

(b) @BRKPT FILE

No periods may be used in the 'FILE' fields.

Format (a) is used to close out the currently active print or punch file, and start a new one. If the file being closed is a standard print or punch file, it is automatically queued for output at an appropriate peripheral device.

If the new file being started is to be a standard print or punch file, a user-defined FILE is not specified. In batch mode, many print or punch files may be opened or closed in succession, by a series of @BRKPT commands. Whenever this is done, a @MSG statement must be used to inform the operator of the number of outputs to be printed.

If a 'FILE' field is specified on a @BRKPT statement, it may only be a 1-12 character name. A @USE statement may be used to attach this internal name to an external filename.

@ASG,URG is normally used to set up the FILE. After it is completed, @BRKPT, @FREE, and @SYM (in that order) must be used to queue it for printing or punching.

Format (b) is used to close out an alternate print or punch file. The 'FILE' field contains the file name of the alternate symbiont file being closed out.

4.4.7.1. Examples of the @BRKPT Statement

```
@BRKPT  PRINT$
```

The current print (either system- or user-defined) file is closed and a new standard print file is opened.

```
@BRKPT  PRINT$/MYPRINT
```

The current print file (either system- or user-defined) is closed and succeeding print will be placed in the file MYPRINT. MYPRINT must be assigned to the run.

```
@BRKPT  PRTFIL
```

The alternate print file PRTFIL is closed. There is no effect upon the current print file.

See the chapter on Sample Deck Setups for examples of @BRKPT used in complete runstreams.

4.4.8. The @SYM Statement

The @SYM statement provides the user with the capability of selecting a symbiont, or class of symbionts, to print or punch selected files. A standard system procedure exists for printing and punching those files produced with the interface routines PRINT$, PRNTA$, PUNCH$ and/or PNCHA$ (standard symbiont files) during the course of a run if they reside in a system-defined mass storage file. As these files are completed, they are entered into the appropriate print or punch queue determined by the run's associated input source. When a @SYM statement is encountered the specified file is entered into the specified symbiont queue.

The format of the @SYM statement is

```
@SYM,OPTIONS  FILENAME,,SYMBIONT
```

The 'OPTIONS' field may contain:

U - Do not decatalogue FILENAME after printing or punching. If U is not specified, FILENAME will be deleted.

C - Used with a punch file when the symbiont field indicates a remote site.

The 'FILENAME' field is used to specify the file to be processed, which should be a Fastrand file.

The 'SYMBIONT' field is the name of a symbiont, or symbiont group, which is to output the file. If omitted, the symbiont associated with the run initiation device is assumed. To transmit a print file to a remote batch site, the site id must replace the symbiont name. @SYM to a demand terminal is not allowed.

Georgia Tech has three onsite print devices and two onsite punch devices. These are:

Symbiont Device Name	Description
PR1	0755 High-speed printer - up to 1100 lpm
PR2	1004 printer - up to 600 lpm
PR3	1004 printer
CP1	1004 punch - up to 600 cpm - with paper tape punch
CP2	1004 punch - no paper tape punch

Output may be directed to any of these devices by specifying the appropriate device name in the 'SYMBIONT' field. Alternatively, one of the following symbiont groups may be specified:

Symbiont Group	Device Selection
PR	PR1, PR2, PR3
PRB	PR1, PR3, PR2
PR23	PR2, PR3
CP	CP1, CP2
CPB	CP2, CP1

When a symbiont group is specified, output is sent to the first available device in the device selection list, searching in the order specified above.

Multiple file printings from mass-storage will be executed concurrently, if possible, for each @SYM statement encountered.

4.4.8.1. Use of @SYM with PRINT$ and PUNCH$

Each run entered into the system has symbionts defined for processing the system initiated print (PRINT$) file and punch (PUNCH$) file. These output symbionts are classified for each run at system generation time. However, it may become necessary to redefine either, or both, output symbionts for a particular run to process all, or portions, of the output file. The 'FILENAME' field is used to denote the print or punch file with either PRINT$ or PUNCH$ respectively. The symbiont field is used as defined above.

For example, runs punching paper tape must insure that their standard punch file is processed by CP1, since CP2 has no paper tape punch. This is accomplished by including

```
@SYM  PUNCH$,,CP1
```

in the runstream.

PR1 has the peculiarity that the character ≠ (octal 077) is used to indicate end-of-line, and is never printed. Thus if a run is to print '≠' characters, the output must be printed by PR2 or PR3. This is done by including

```
@SYM  PRINT$,,PR23
```

in the runstream.

The device association may be set to send output to a remote batch site from onsite (provided arrangements are made in advance), and vice versa.

Examples of @SYM in complete runstreams are given in the chapter entitled Sample Deck Setups.

4.4.9. The @COL Statement

Each 1004 of the system assumes the 80 columns (026) mode for reading and for punching. The system at Georgia Tech has been modified to allow the user to read cards in the 029 mode. Note that all cards punched by the system are in the 026 code.

4.4.9.1. 026 MODE

The 026 mode is assumed as standard. If a user's run is all 026 code, he does not need any cards for conversion. If part of a user's run is 029 code and he needs to revert to the 026 mode, then the following card is used.

Note: This form of the @COL statement is fixed in format. This form of the @COL card is written:

```
column.......123456789
characters...@COL 6
```

4.4.9.2. 029 MODE

This form of the @COL card allows the user to run an 029 deck or part of a deck.

Note: This form of the @COL statement is fixed in format. This form of the @COL card is written:

```
column.......123456789
characters...@COL 9
```

Note: The system reverts to 026 mode at each @RUN and @COL 6 statement. Any number of these two forms of the @COL statement, @COL 6 and @COL 9 may appear in the control stream and may appear anywhere in the deck following the @PWRD statement.

The @COL 6 and @COL 9 statements are recognized only for onsite batch. Remote batch sites should modify their communications software to be compatible with the keypunches available. Such modifications have been accomplished at a number of sites.

4.4.10. The @PWRD Statement

The @PWRD statement is used in onsite batch and remote batch run decks to specify the user's batch password.

A different batch password is assigned to each authorized user-name. In order to have his run processed, the batch user must specify his batch password on a @PWRD statement.

The purpose of passwords is to prevent use of a reference-number/user-name combination by an unauthorized person. Thus, passwords are useless unless each user keeps his password secret. The computer center recommends that when punching a @PWRD statement, the keypunch print function be turned off. In case of accidental disclosure of a password, a new password may be assigned by contacting your Departmental Computer Coordinator. The system never prints passwords on users' output.

Batch passwords consist of six alphanumeric characters, the first two of which are an abbreviation for the user's department.

4.4.10.1. Format and Placement of the @PWRD Statement

The format of the @PWRD statement is fixed and it must be punched as follows:

column:	1	2	3	4	5	6	7	8	9	10	11	12
contents:	@	P	W	R	D		X	X	X	X	X	X

where XXXXXX represents the assigned password. Columns 13-80 are ignored by the system.

The @PWRD statement must immediately follow the @RUN statement in each batch run submitted.

4.4.11. The @ BIN Statement

The @ BIN statement is used to identify the bin at the input/output counter to which a batch run's output is to be returned. Normally, @ BIN statements are supplied at the I/O counter.

A @ BIN statement with blanks in the bin number field causes the run following it to be assigned to the same bin as the run preceding it. This form of @ BIN statement must be used by the user when submitting sequenced runs (see the @RUN statement--S option).

A @ BIN statement with other than blanks in the bin number field should not be used except when a permanent bin has been assigned to the user for batch work.

The format of the @ BIN statement is fixed as follows:

column:	1	2	3	4	5	6	7	8	9	10	11	12
contents:	@		B	I	N				X	X	X	X

where XXXX is the desired bin number, or blanks for a null bin card. Columns 13-80 are ignored by the system.

4.4.12. The @LOG Statement

The @LOG statement has the format

```
@LOG          text
```

where 'text' may be any characters, except as follows. The semicolon (;) is used as a continuation character; therefore it cannot be part of the text. The character sequence space-period-space (.) is not allowed as part of the text because this sequence denotes the start of the comment field.

The 'text' will be printed at the end of the user's output, along with other log and accounting information.

4.5. Input/Output Specification Statements

4.5.1. General File Information

4.5.1.1. Introduction

The Exec 8 Operating System provides modern Input/Output capabilities designed for the time-sharing environment, including

(1) Dynamic allocation and release of mass storage space.

(2) Automatic scheduling and selection of facilities such as tape drives.

(3) Protection from undesired tampering with private data and programs.

(4) Exclusive use provisions providing synchronization of accesses to information.

(5) Optional cycling of versions of data and programs, allowing "backing up".

(6) A filename system that is simple for simple runs, but generalizes for functions such as library files, etc.

(7) Support of a wide variety of input/output peripherals, varying in speed and capacity.

This section on General File Information introduces file concepts. The sections on the @ASG, @MODE, @CAT, @FREE, @USE, and @QUAL statements provide detailed information allowing file structure manipulation. Manipulation of file contents is performed by the system processors, utility routines, and user programs.

4.5.1.2. Input/Output Peripheral Equipment

The information in this section is given to (1) acquaint the user with the various I/O peripherals available, (2) allow the user to compare the relative advantages and disadvantages of the different devices, and (3) give the user background to properly code the equipment 'TYPE' subfields on I/O specification statements.

This section deals only with mass storage and magnetic tape devices. Paper and other peripherals are not generally referenced by users in I/O specification statements.

4.5.1.2.1. Mass Storage Equipment

The following table describes the mass storage equipment available in the Georgia Tech configuration.

Mass Storage Equipment

Type	Number of Units	Storage Capacity (words/unit)	Access Time min.	Access Time avg.	Access Time max.	Rotation Speed (rpm)	Maximum transfer rate (words/sec)
FH 432	3	262,144	120 μs	4.3 ms	8.5 ms	7120	240,000
FH 880	3	786,432	160 μs	17 ms	34 ms	1770	60,000
FH 1782	1	2,097,152	120 μs	17 ms	34 ms	1770	240,000
FASTRAND II	2	22,020,096	1 ms	92 ms	156 ms	870	30,566*

*Continuous transfers at this rate are not possible. Effective rate is roughly 25,590 words/sec.

Note: Each 1108 word contains 36 bits which may be considered as six six-bit characters.

As this table indicates, our configuration provides a considerable amount of mass storage space with a variety of access times and transfer rates. The smaller capacity, higher speed FH 432, FH 880, and FH 1782 drums are more efficiently utilized for performing various functions of the Executive and for the assignment of temporary user files. The much larger, but slower, Fastrand II drums are more efficiently utilized for permanent, catalogued, storage of user files that are moderate in size and are frequently needed.

Fastrand drum units are logically divided into positions, tracks, and sectors. A sector is 28 36-bit words in length, and is the smallest addressable increment of Fastrand storage. A track is 64 sectors, or 1792 words, and corresponds logically to the path made by a read/write data head as the drum makes one revolution. A position is 64 tracks, or 114,688 words, and corresponds logically to the paths made by all of the 64 read/write data heads with the "boom" fixed in one of its 192 positions.

The FH series drums are word-addressable and are not physically organized like Fastrand. However, Exec 8 is equipped to simulate Fastrand on these drums, providing the capability to run programs written to use Fastrand on the FH drums, increasing throughput dramatically in some cases. In addition, this allows the executive to dynamically allocate any drum type to a program and still insure proper functioning of that program. The FH series drums may also be used as word-addressable devices.

4.5.1.2.2. Magnetic Tape Equipment

The Georgia Tech configuration includes one Uniservo 8C Magnetic Tape Subsystem. The subsystem includes 8 tape units, each with the following specifications.

Tape Handling Speed	-	120 inches/second
Rewind Speed	-	240 inches/second
Rewind Time	-	2.0 minutes for a 2400 foot reel
Recording Densities (frames/inch)	-	800, 556, and 200
Transfer Rates (frames/second)	-	96,000 @800fpi; 66,666 @556 fpi; 24,000 @200 fpi
Recording Format	-	7-track, even or odd parity
Features	-	optional hardware character translate

4.5.1.3. Temporary versus Catalogued Files

The normal user will be concerned with files on two different media--tape and mass storage. A tape file is always temporary, meaning that the executive retains no record of the assignment after run termination. The physical tape reel itself may be either a scratch tape to be scratched, a scratch tape to be saved, or a tape previously saved (see Section 4.5.3.4). A mass storage (drum or Fastrand) file may be either temporary or catalogued. A temporary mass storage file may be used for either random or sequential access throughout a run. At run termination the executive returns the mass storage space occupied by the file to the available pool and retains no record of the assignment (other than log entries showing the space used). A catalogued mass storage file may be used in the same manner as a temporary mass storage file; however after the run terminates, the executive maintains the file name and location of the text of the file in its directory, and the space occupied by the file is marked as not available for allocation to any other file. Thus the information in the catalogued mass storage file may be referenced and/or updated in a subsequent run.

4.5.1.4. Notation for Filenames

The syntax of filename is

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

where brackets enclose syntactically optional fields.

Qualifier and filenames are 1-12 alphanumeric characters ("$" and "-" are also allowed). Keys have 1-6 characters from the entire Fieldata character set, excluding only space, comma, slash, period, and semicolon. F-cycles are numbered upward from 1, and may be preceded by a "+" or "-" in some cases.

When the qualifier is omitted, the user-name from the @RUN control statement is used, except in the special case where a leading asterisk appears before the filename and a qualifier has been previously furnished on a @QUAL statement. When the F-cycle is omitted, the most recently created F-cycle is assumed.

For temporary files (both tape and mass storage), the F-cycle, read-key and write-key are meaningless. A qualifier may be used, but in general there is no need to do so.

For catalogued mass storage files, the qualifier provides 24-character uniqueness in the directory. In addition, if a user catalogues files without specifying a qualifier, his user-name is used and the filename will be unique as long as the file designator is unique for that user. The F-cycle number serves to maintain successive versions of the same file (same qualifier and file).

F-cycles

As stated earlier, F-cycles apply only to catalogued mass storage files. For a given file and qualifier, say MYNAME*CATFIL, there may be between 1 and 32 associated F-cycles, denoted MYNAME*CATFIL(1), MYNAME*CATFIL(2),..., MYNAME*CATFIL(32). The F-cycles of a file and qualifier have some attributes of being the same file, and some of being distinct files. For instance, all F-cycles of a file must have the same read and/or write keys, if such exist. However, although it may be logically inconsistent to do so, different F-cycles of a file need not contain data that is related in any way (they may even be of entirely different format).

A simple cataloguing action creates absolute F-cycle 1 of a file. To create the next F-cycle, '+1' must be specified in the F-cycle field of an @ASG,C, @ASG,U, or @CAT. (A catalogue request specifying an unsigned or negative integer will be rejected.) This file is given an absolute F-cycle one higher than previously existed. Thus the most recently catalogued F-cycle is the one with the highest absolute number, and is called the current F-cycle.

To assign the current F-cycle of a file, the F-cycle field may be void, contain '+0', or contain an unsigned integer that is the absolute F-cycle of the file. F-cycles older than the current one may be referenced either by their absolute F-cycle or by relative F-cycle. An F-cycle designation of '-1' refers to the next-to-current F-cycle; '-2' refers to the F-cycle created before that; and so on.

Suppose absolute F-cycles 1 through 3 of MY*FILE are currently catalogued. Then the following designations are equivalent:

MY*FILE and MY*FILE(+0) and MY*FILE(3)
MY*FILE(-1) and MY*FILE(2)
MY*FILE(-2) and MY*FILE(1)

If MY*FILE(+1) is subsequently catalogued, then the following equivalency holds:

MY*FILE and MY*FILE(+0) and MY*FILE(4)
MY*FILE(-1) and MY*FILE(3)
MY*FILE(-2) and MY*FILE(2)
MY*FILE(-3) and MY*FILE(1)

Users are cautioned not to catalogue more than 32 F-cycles of a file.

4.5.1.5. Mass Storage Policies and Procedures

Following sections will describe the capabilities available for manipulating mass storage files. However, abuse of these capabilities, either knowingly or unwittingly, will degrade system performance for all users. Therefore, it is requested that all users conform to the following guidelines.

a) Use the higher speed, FH series drums only for temporary mass storage files needed within a particular run. That is, do not catalogue any files with an equipment type other than F2.

b) Use the Fastrands for catalogued files that are needed frequently.

c) Do not leave very small files catalogued on Fastrand; use punched cards or punched paper tape instead.

d) Do not leave very large files catalogued on Fastrand; use magnetic tape instead.

e) Perform housekeeping chores regularly and frequently; delete files no longer needed; reduce the size of files if possible; store large files or collections of smaller files on magnetic tape when they are infrequently needed, etc. Sections in this manual relevant to file housekeeping include 6.7, the Listfiles Processor; 5.4.8, the @PACK Statement; and 5.4.5, the @PRT Statement.

The Rich Electronic Computer Center follows procedures that attempt to guarantee the presence of user files during hours of operations. Periodically, all user catalogued Fastrand files are copied to magnetic tape. Whenever it becomes necessary, due to unanticipated failures or due to anticipated preventive maintenance, the latest copy is reloaded.

In conjunction with the guidelines above, the following procedures are in effect to help prevent abuse of mass storage.

The computer center has established (1) a maximum allowable file size, and (2) a file expiration period. Contact the Office of the Director or your Departmental Computer Coordinator for current information on these parameters. At any time, the computer center may delete files

(a) catalogued with equipment type other than F2

(b) of zero size

(c) of size greater than the maximum allowable size, unless catalogued with the G option

(d) catalogued by users not in the current account file

(e) files that have not been assigned for a period longer than the expiration period.

Additional categories of files may be designated to be deleted in the future. The Director, Rich Electronic Computer, as well as all departmental computer coordinators, will have current information concerning mass storage policy.

Each user is responsible for recreating his own files if it should become necessary. However, if an important file is deleted and cannot be conveniently recreated by the user, the computer center may be able to help the user by reloading the file from a backup tape. In such a case a Programmer Aide should be consulted.

4.5.1.6. Definition of "Assigned"

Each run active in the system has associated with it a program control table, or PCT. This table is used by the executive to control handling of the run. The PCT may not be altered by the user.

When an input/output request for a file is made by an activity of a run, the I/O portion of the executive must have detailed information about the file immediately available. Thus, one main use of the PCT is to hold detailed information about all files that may be referenced in I/O requests. In addition, peripheral units may have to be conditioned before I/O can take place (e.g., a tape unit may have to be dedicated to the file).

Thus, when a file is assigned, the run's PCT and the computer's peripheral units are set up to allow efficient I/O to take place.

On an assignment request for a temporary file or a file to be catalogued, all information to set up the PCT and peripheral units is obtained from the assign request (@ASG statement or equivalent), system standard, and configuration constants. On an assignment request for a previously catalogued mass storage file, information from the directory supplements the above sources. The directory information is created when the file is originally catalogued, and may be updated whenever the file is changed.

4.5.1.7. Methods of Assignment of Files

The assignment of a file may occur in one of three ways:

(1) Via an @ASG control statement.

(2) From within the executive itself as a result of an @XQT, @START, or processor call statement.

(3) Via an executive request from within a user program or system processor.

The user is always free to assign a file via an @ASG statement. If this is not done, case (2) or (3) may apply.

The user must always explicitly assign a tape file, either via an @ASG statement (the preferred method), or via an executive request from within a program. Such internal requests may be made, for example, by appropriately calling the FORTRAN library subprogram, ERTRAN. The user must explicitly assign a temporary mass storage file, except that if a write is requested on an unassigned file from ALGOL or FORTRAN, the WRITE routine (which, as far as the executive is concerned, is part of the user's program) will automatically make an executive request to assign a temporary mass storage file to the run.

If a file is referenced on an @XQT or @START statement, and the specified file is not assigned, the executive will attempt to assign the file from its directory. If the filename is not a catalogued mass storage file, an error is indicated. Files assigned in this manner are not automatically freed (de-assigned).

Ordinarily, system processors reside in the catalogued mass storage file SYS$*LIB$, which is always assigned to every run. However, if a processor resides in another file, the filename is given in the command field of the processor call statement; e.g.,

```
@SYSTEM*ALTPRO.FOR,IS  PF.MAIN
```

If such a file is not assigned, it is assigned by the executive exactly as for @XQT and @START.

Files named in the specifications on any type of processor call (PF in the above example) are not handled by the executive proper but by the processor called. Again, if the files concerned are not assigned, the processor attempts an assign out of the directory. Following use of such files, the processor returns the file to the assign status it had when the processor received control. Files named in MAP source language statements and on @ADD control statements are handled identically to files named in the specifications fields of processor calls.

4.5.1.8. Unloaded Mass Storage Files

Since the available mass storage space is not infinite, it is possible for it to be exhausted unless special action is taken. To prevent mass storage overflow, the executive will automatically unload selected catalogued files. These files will have their text written to magnetic tape (unless a current copy already exists on a backup tape), and the space occupied by the text of the file is released to the available pool. The location of the text is noted in the directory.

4.5.1.8.1. Selection of Files to Unload

The selection of files to unload is based on the Unload Eligibility Factor (UEF). When mass storage availability becomes critically low, UEF's are computed for all files in the system, and those with the largest UEF's are unloaded until mass storage availability returns to normal.

The most important factor in computing the UEF is the time since last assignment. The theory is that files not referenced for a long period of time will not be referenced again soon; hence they are given a large UEF to start with.

A value is added to the UEF based on the average time between assignments, computed as the current date and time less the cataloguing date and time, all divided by the number of assigns. The value added to the UEF increases as the average time between assignments increases, so that frequently used files have less chance of being unloaded.

It is desirable to minimize the number of unloaded files, so a bias is added to the UEF based on the size of the file. The larger the file, the larger the size bias.

If a file is outdated by a more recent F-cycle, its UEF is increased, because outdated F-cycles are not referenced as often as current F-cycles.

Finally, a private file will have its UEF increased, since unloading a public file will potentially inconvenience more people.

4.5.1.8.2. Reloading of Unloaded Files

When an unloaded file is assigned by the user, the exec automatically initiates a reload of the text from tape. If the assignment was via an @ASG control statement, the run is held ("WAITING ON FACILITIES" is printed for demand runs). When the reload is complete, the hold is released ("READY is printed for demand).

If the assignment was by some other method (see 4.5.1.7), the reject status 400003000000 is returned, usually causing undesirable results. Since all user catalogued files are subject to being unloaded, an @ASG,A control statement should be used to assign the file (and reload it if it is unloaded) prior to referencing the file in any other way.

4.5.1.9. Disabled Files

Whenever the system encounters an abnormal situation manipulating user files, it will mark the file "disabled." The user will be notified that a file is disabled by a FACILITY REJECTED or a FACILITY WARNING message upon the next attempt to assign the file. The FURPUR command, @ENABLE, may be used to clear disable flags for the file; however, this may not correct the condition causing the disable.

4.5.1.9.1. Incomplete Write Disable

The Incomplete Write Disable will be set by the system if (1) the file was assigned when a system failure occurred, (2) the file is not read only, and (3) the file was not catalogued with the G option. The message FACILITY WARNING 000000000200 will be issued when the file is assigned, and the run will continue. This indicates that a user run that may have been changing the file did not complete due to a system failure; hence the contents of the file are questionable. After examining the file, the user may always clear the Incomplete Write Disable with an @ENABLE command.

4.5.1.9.2. Destroyed Disable

The Destroyed Disable will be set by the system if a hardware or software error has malformed the directory entries for the file. The message FACILITY REJECTED 400000000400 will be issued on the next assign attempt, and the run will be terminated unless demand. Recovery may be attempted by using the @ENABLE command, but complete recovery is unlikely. If necessary, contact a member of the Computer Center staff to reload the latest backup of the file.

4.5.1.9.3. Bad Backup Disable

The Bad Backup Disable will be set by the system if (1) a tape error occurred while attempting to load the text of a file from a backup tape, or (2) a hardware or software error caused the loss of the backup information (e.g., reel number of the backup tape) from the directory. The message FACILITY REJECTED 400000000100 will be issued on an assign attempt for the file, and the run will be terminated unless demand. Recovery may be attempted via the @ENABLE command. If necessary, contact a member of the Computer Center staff to reload the file from the latest good backup tape.

4.5.2. The Mass Storage @ASG Statement

The general form of the mass storage @ASG statement is

@ASG,OPTIONS FILENAME,TYPE/RESERVE/GRANULE/MAXIMUM

The fields of the statement are explained in succeeding paragraphs and in the order of appearance on the statement.

4.5.2.1. The 'OPTIONS' Subfield

The 'OPTIONS' subfield is used to cause a file to be catalogued (or decatalogued) and to place or remove constraints on the use of the file. It should be noted that when an error condition occurs which would cause a batch run to be terminated, the demand user receives an error message and is allowed to submit a new statement.

Cataloguing options are as follows:

C - Specifies that the file is to be catalogued if the run terminates normally. If a @FREE command (control statement or executive request) is encountered for the file prior to termination, the file is catalogued at that time (see 'the @FREE statement'). If a file by this name already exists in the master directory, the run is placed in the error mode.

U - Same as 'C' option except that the file is to be catalogued at run termination regardless of the manner of termination (beyond this statement). The @FREE command may cause cataloguing prior to the termination.

R - Specifies that the file is to be placed in the "read-only" state when it is catalogued. This option is meaningful only when the 'C' or 'U' option is also present. The file can only be read or decatalogued. Any activity requesting to write in the file will be placed in the error mode.

P - Specifies that the file is to be catalogued as a "public" file rather than a "private" file. The distinction between them is that only the runs which have the same user-name as the run which created the file can access a "private" file while any run can access a "public" file. (For privacy in "private" files, see the discussion concerning the two 'KEY' subfields.)

W - Specifies that the file is to be catalogued as a write only file. The file can only be written into, and in the process extended.

G - Specifies that the catalogued file is not to be saved by the computer center on backup tapes. The file will disappear whenever a mass storage initialization is performed by the operators.

V - Specifies that the catalogued file is never to be "rolled out." If the V option is not specified, a file may be rolled out to provide adequate working space on Fastrand. If a rolled out file is subsequently assigned, the run will be held and the file automatically rolled in. The V option should be used only for special purpose files that must never be rolled out, such as those used by a real-time program.

The above options are for use only with files that are not presently catalogued. If neither of the cataloguing options ('C' and 'U') appear, the file, unless currently catalogued, is treated as temporary and released at run termination. It will be released prior to run termination if a @FREE statement is encountered. In the absence of the 'P' option, a file is always catalogued as "private".

Options to be used when the @ASG statement names a file that is presently catalogued are as follows:

D - Specifies that the catalogued file is to be deleted from the directory (decatalogued) if the run terminates normally or when a @FREE command is encountered prior to termination. The executive will insure the file is assigned only to this run at the time of release.

K - Same as 'D' option except that the file is to be deleted at run termination regardless of the manner of termination. The @FREE command may cause the file to be decatalogued prior to termination.

X - Specifies that this run is to have "exclusive use" of the file until the run has terminated or the file is released via the @FREE command. No other run can be using the file. If the file is not currently catalogued, the 'X' option is not needed because the run necessarily has "exclusive use". This option is ignored for files catalogued with the R option.

A - Specifies that the file is currently catalogued and insures that the executive will not treat the file as temporary if the name cannot be found. The run will be terminated if the name cannot be found in the directory.

The above options are to be used only with files that are currently catalogued. If neither of the decataloguing options ('D' or 'K') appear, the catalogued file is left intact at run termination. If either the 'D' or 'K' options appear and the file has either or both keys, the key(s) must be specified. Failure to do so causes the run to be placed in the error mode.

An option to be used for a temporary file (not catalogued and not to be catalogued) is as follows:

T - Specifies that the file is temporary and allows it to have a name the same as that of a catalogued file. No thought need be given as to whether a file by this name is currently catalogued. If this option is <u>not</u> present for temporary files, the system will attempt to find the file in the directory. If a find is made, the assignment will be made from the directory.

The following options control the dumping of catalogued mass storage files at a checkpoint, and subsequent system action on restarting:

B - Dump the file as a part of any checkpoint.

E - Reload this file if any other run has referenced the file since checkpoint.

H - Reload this 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.

N - Rename this file upon reloading if a catalogued file with this name exists.

Option B forces the file to be dumped on a checkpoint. Without one of the options, E or H, the file is always reloaded on restart. Options M and N control the manner of reload.

4.5.2.2. The 'FILENAME' Field

The field 'FILENAME' on the @ASG statement is used to specify the external name of the file. The name must be present and is specified in the normal manner:

QUALIFIER*FILE(F-CYCLE)/READ-KEY/WRITE-KEY

where the 'QUALIFIER' and '*' are optional and neither the 'QUALIFIER' nor the 'FILE' may exceed 12 characters. The 'F-CYCLE' number may need to be specified for catalogued files.

4.5.2.2.1. The 'READ-KEY' and 'WRITE-KEY' Subfields

When cataloguing, the subfields 'READ-KEY' and 'WRITE-KEY' lock a file against indiscriminate reading and writing, respectively, by other users. They may contain up to six characters and all characters are legal except the blank, the slash, the comma, the period, and the semicolon. A file is catalogued with 'READ' and/or 'WRITE' lock by specifying the 'READ-KEY' and/or 'WRITE-KEY' subfields along with the 'C' or 'U' option. To gain read and/or write access to such a file, the appropriate key(s) must be specified at assign time or the request(s) will not be honored. (Once the assignment has been made, with the appropriate key(s) made available through the @ASG or @USE statement, the key(s) need not be specified in further references.)

A combination of the two keys is used for cataloguing. The following table shows the action allowed according to the key(s) given at cataloguing time and the key(s) given at assign time. Where "message" appears as an action, a 'FAC WARNING DDDDDDDDDDDD' message will be printed.

If a key is furnished and it does not match the catalogued key, the run is aborted, and the message will be 'FAC REJECTED DDDDDDDDDDDD'.

	Key(s) Specified at Assign Time			
Key(s) Specified at Cataloguing Time	Read	Write	Both	Neither
Read	Read Write	Abort	Abort	Write
Write	Abort	Read Write	Abort	Read
Both	Read Message	Write Message	Read Write	Message
Neither	Abort	Abort	Abort	Read Write

4.5.2.3. The Facilities Field

On all @ASG statements (mass storage, magnetic tape), the field that follows the name field is called the 'FACILITIES' field. As shown previously, the facilities field for the mass storage @ASG statement is

'TYPE/RESERVE/GRANULE/MAXIMUM'

In general, if the file is catalogued and to be read, the entire facilities field need not be specified.

The subfield 'TYPE' specifies that the statement applies to mass storage and, in addition, points out the type of equipment to be used.

Fastrand Format (Simulated on FH Drums)

Type	Equipment Used (in order of preference)
F4	FH 432, FH 880, FH 1782, FASTRAND II
F8	FH 880, FH 1782, FASTRAND II
F17	FH 1782, FASTRAND II
F	same as F4
omitted	same as F4
F2	FASTRAND II

Word-Addressable Format

Type	Equipment Used (in order of preference)
D	FH 432, FH 880, FH 1782
D4	same as D
D8	FH 880, FH 1782
D17	FH 1782

The subfield 'RESERVE' is used to specify the approximate number of granules to be used by the file. The subfield 'GRANULE' is used to specify the granule size. In certain cases, either or both subfields may be omitted. If the granule subfield is specified, it must contain either 'TRK' for track granularity, or 'POS' for position granularity. Unless a file is larger than roughly a thousand tracks, track granularity should be used. For very large files, specifying position granularity will save space in internal executive tables. If the granule specification is omitted, the granule is assumed to be 'TRK'. The granule subfield is ignored if the file is currently catalogued.

The reserve subfield is ignored and need not be specified when the file is catalogued and is to be read only. If the file is to be created or updated, the reserve may contain an integer specifying the number of granules to reserve for the file (on an update the reserve specification includes that portion of the file that already exists). If the reserve specification is omitted, no granules (or additional granules) are initially assigned; they are assigned dynamically as needed. When the reserve is supplied but exceeded, additional granules are also assigned dynamically as needed.

Note: When creating a file, the reserve subfield should contain a reasonable estimate of the number of granules needed. If a file can be contained within the limits of the reserve, the run is assured of being able to create the file without delay. In addition, the specification of a reserve aids the executive in allocating Fastrand area efficiently. (If a reserve is used, the tracks will be adjacent, if possible.)

If the file takes fewer granules than reserved, the empty granules are returned to the available status when the file is catalogued. The reserve value is placed in the directory and will be used on future updates unless a reserve is supplied on the update @ASG statement. In that case, it is used and replaces the previous value in the directory.

The subfield 'MAXIMUM' is used to indicate that the run is to be terminated if the length of the file being created or updated exceeds the number of granules specified. This field is used primarily to insure that a run-away-file situation does not occur during debugging. However, it may also be used to override the system-maximum for all files (128 tracks). The maximum subfield is never a required specification. If the file is being created or updated and a maximum is given, its value is placed in the directory along with the name, type, reserve, and granule size.

If a maximum was supplied when the file was catalogued, its value is retained and used when an update occurs. If a maximum is supplied on the updating @ASG statement, it is used. It is also placed in the directory, thereby replacing the previous maximum.

Although space for word-addressable format (types beginning with D) is physically allocated in granules of the 'TRK' or 'POS' size specified, the 'RESERVE' and 'MAXIMUM' must be stated in number of words, rather than in number of granules.

4.5.2.4. Exclusive Use and Facility Handling

The Exec provides for the placement of @ASG and @FREE statements anywhere within the control stream. Dynamic assign and free requests may appear within the programs. These features allow the user to assign and free files as required, without "tying-up" the files and/or 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) Exclusive use of a catalogued file that is being used by another run.

(3) Use of a catalogued file that is assigned exclusively to 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 control stream, the Executive:

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

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

By placing all magnetic tape and exclusive use requests before the first task of a batch run, the user will be assured that the run will not open until all facility requirements can be met, and hence the run will be processed without delay once it starts.

Should an @ASG statement be encountered which cannot be satisfied due to assignment of the facilities or file to another run, the run will be held in wait status until the @ASG can be satisfied. In some cases, this may cause an infinite wait. The operator will terminate runs waiting for excessive periods of time unless it is obvious that the wait will be eventually terminated. An example of this is a @START run attempting an @ASG,AX of the same file it is in. Since that file is assigned to the exec for reading the runstream, the @ASG,AX can never be satisfied.

4.5.2.5. Examples of the Mass Storage @ASG Statement

Consider the following examples of @ASG statements for mass storage.

```
@ASG,CR FILEX,F2/5
```

If the run terminates normally or a @FREE statement for FILEX is processed, FILEX will be catalogued in the "read-only" mode. Five tracks are assigned initially and the system-maximum size is assumed as no maximum was specified.

```
@ASG FILEX
```

The master directory of catalogued files will be searched for FILEX. If a find is made, it will be assigned as per the options and specifications (including equipment type) with which it was originally catalogued. If no find is made, a temporary file named FILEX will be assigned with equipment type F4.

```
@ASG,AK FILEX/A2294B
```

FILEX is currently catalogued and is to be decatalogued at run termination or if a @FREE statement is processed for file FILEX. The key A2294B is required to read and/or decatalogue the file.

```
@ASG,T FILEX,F/4//5
```

FILEX is a temporary file requiring 4 tracks of the fastest drum available. Fastrand format is to be used. Termination is to occur if more than 5 tracks are required.

```
@ASG,CPG FILEX,F2
```

FILEX is to be assigned and will be catalogued on Fastrand II upon normal run completion or a @FREE. The computer center is not to copy the files onto backup tapes. The file is to be public.

```
@ASG,T FILEX,D/30000
```

At least 30,000 words of the fastest word-addressable drum available are to be reserved for the temporary file FILEX. Allocation is by track. Since a track contains 1792 words, 17 tracks or 30,464 words will actually be reserved.

4.5.2.6. Diagnostic Messages

A generalized format is currently used in the print file assigned to each run for the ASG, MODE, CAT, FREE, and USE statements. The format is as follows:

(1) (Statement Image)
FAC REJECTED DDDDDDDDDDDD

(2) (Statement Image)
FAC WARNING DDDDDDDDDDDD

The first message will appear for a run that is aborted due to a statement that cannot be honored by the system. The second message is a warning that the statement could cause a problem. In either case the reason for rejection or warning is determined by examining the bits set in the octal word 'DDDDDDDDDDDD'. The following table defines the meaning of the bits if set (1 = set). Bits are numbered 35-0 reading left to right.

BIT	ACTION	DESCRIPTION OF MEANING IF BIT SET
35	K	Request not accepted - examine rest of bits as to why.
34	K	Field error in statement other than syntax. Also option conflict 'MLH', 'OE', 'IB'.
33	W	Filename has already been assigned to this run.

32	K	File previously catalogued.
31	K	Equipment type on ASG statement is not compatible with catalogued equipment type.
30		Name found in attached name list in PCT.

29	W	12-character name is not unique (that portion of name used as internal name for I/O packets).
28		X (exclusive use) option was already on this file, on an @ASG,X.
27	K	Read key incorrect for catalogued file.

26	K	Write key incorrect for catalogued file.
25	W	Write key exists in directory, not specified on ASG statement (assigned read mode only).

BIT	ACTION	DESCRIPTION OF MEANING IF BIT SET
24	W	Read key exists in directory, not specified on ASG statement (assigned write mode only).
23	K	Read key furnished on ASG statement; none exists in directory.
22	K	Write key furnished on ASG statement; none exists in directory.
21	K	'A' option specified on ASG statement and filename was not found in directory.
20	K	Invalid reel number on ASG statement for catalogued tape file.
19		Mass storage file has been rolled out. Loading of file initiated.
18		Request on wait status for facilities.
17	K	Option conflict for catalogued file, both 'D' and 'K' or 'CUPRW' which are options for new files.
16		File is assigned exclusively to some other run.
15		File already assigned to another run. (This is cause for rejection of an @ASG,X.)
14		Find made in directory and not assigned.
13	K	User name incorrect for catalogued private file.
12		Equipment type is tape.
11		Read only file catalogued with 'R' option.
10		Write only file catalogued with 'W' option.
9	K	Equipment is down.
8	K	File has been destroyed due to hardware errors or loss of directory information. (See note 2.)
7	W	File was assigned with write enabled at the time of a system failure. Attempted writes may not have been completed. (See note 3.)
6	K	File has been lost due to a backup tape error or loss of backup information. (See note 2.)
5		Unused.
..		
0		

Note 1: K in ACTION means that the run will be terminated, unless it is demand. W in ACTION means that a warning message will be printed.

Note 2: Assignments rejected due to "destroyed" or "bad backup" conditions (bits 8 and 6) may be further processed if an @ENABLE command is given for the file. This does not imply, however, that the condition causing the original reject will be corrected by the @ENABLE.

Note 3: After insuring that the file is intact, the user may remove the "incomplete write" warning (bit 7) via the @ENABLE command.

Examples of Diagnostic Messages

```
FAC REJECTED  400010000000
     Bits set: 35,21
```

The file named on the @ASG,A was not catalogued. It may have been deleted by the computer center due to (a) file size 0, (b) file size greater than allowable maximum, (c) file had G-option and mass storage was reinitialized, (d) file had not been referenced for the expiration period. Another common problem is that the file was catalogued under another user-name and no qualifier was specified in either run. A qualifier of the original user-name must be added to all references of the file in the current run.

```
FAC REJECTED  400000020000
     Bits set: 35,13
```

The file was catalogued private under some other user-name. The file may not be referenced except from runs bearing that user-name.

```
FAC REJECTED  400000400000
     Bits set: 35,17
```

An attempt was made to catalogue a file that is already catalogued.

```
FAC WARNING  100000000000
     Bit set: 33
```

An assign was requested for a filename that was previously assigned in this run. (This may have occurred via an @ASG statement, by reference on a @START, @XQT, or processor call statement, or from within a user program or system processor.) If the same file was intended on both @ASG's, no action is necessary. If different files were intended, the first must be @FREE'd before the second is assigned. (Note that in this case, at least one of the files must be temporary.) If the status of assignment is to be changed

by adding keys or altering the initial reserve or maximum, the file must be @FREE'd and reassigned.

```
FAC WARNING 000300000000

     Bits set:  25,24
```

Both read and write keys exist in the directory, but neither were specified on the @ASG. The file must be @FREE'd and reassigned with key(s) to permit any input and/or output to take place.

4.5.3. The Magnetic Tape @ASG Statement

For magnetic tape the format of the @ASG statement is:

```
@ASG,OPTIONS FILENAME,TYPE/UNITS/LOG/NOISE,REEL1/REEL2.../REELN
```

4.5.3.1. The 'OPTIONS' Subfield

As explained in section 4.5.1.3., a tape file should always be temporary. In actuality, tape files may be catalogued similarly to mass storage files, but this leads to many difficulties and, except in rare cases, is of no real value. Thus, users should never use the C or U options on a magnetic tape @ASG statement.

The option to specify a file as temporary is:

T - same as for mass storage

The following options, called the 'MODE OPTIONS', correspond to the 'MODES' available with the 'SET MODE' function of the magnetic tape handler:

L - Low density (200)
M - Medium density (556)
H - High density (800 - assumed)
E - Even parity
O - Odd parity (assumed)
B - Binary (no translate - assumed)
I - Decimal (translate)

Note: If the I option is specified, characters are translated by hardware as they are read from tape according to the table in Appendix C.

4.5.3.2. The 'FILENAME' Field

The 'FILENAME' field is the same as for the mass storage @ASG statement, and is required. It is doubtful that specifying a qualifier would be of any advantage. F-cycle, read-key, and write-key should not be specified, since a tape file is temporary.

4.5.3.3. The Facilities Field

The field 'TYPE/UNITS/LOG/NOISE' is called the "facilities" field.

The subfield 'TYPE' is used to show that the @ASG statement is for magnetic tape and contains a symbol denoting the exact type of tape units required. This specification is required. Exec 8 allows many different tape equipment type specifications, owing to the large number of possible configurations. However, since only one equipment type exists in the Georgia Tech configuration, only one type need be described here. That type and its meaning is as follows:

T	TAPE, TYPE INDEPENDENT

The system does not assume the translate option 'I' simply because the hardware translation feature is available. This action must be called for either on the @ASG statement or by use of the 'SET MODE' function of the magnetic tape handler.

The subfield 'UNITS' is an integer (1 or 2) specifying the number of units required. If omitted or an integer other than 1 or 2, the number of units is assumed to be one. A maximum of 2 units per file is allowed. Specification of 2 units for multiple reel tape files will allow rewinding of one reel while processing another.

The subfield 'LOG' is used in configurations containing more than one tape channel to balance I/O activity between channels. Since Georgia Tech has only one tape channel, this subfield is meaningless and need never be specified.

The 'NOISE' subfield is an integer in the range 1-99 that will override the system standard (18) if present. The system standard noise constant is assumed if the 'NOISE' subfield is not specified. Any physical tape block of length less than the noise constant (in characters) will be disregarded. If noise blocks are encountered, a message will be placed at the end of the run's print file.

4.5.3.4. The Reel Field

The field 'REEL1/.../REELN' is called the reel field and is used to specify the physical reels to mount for the file. If the reel field is omitted, the operator will be requested to mount a scratch reel. If the file is contained on a single reel, the reel number is specified and the operator is requested to mount that reel. If the file extends over two or more reels, the reel numbers are specified in order, separated by slashes. The operator is requested to mount each reel at the proper time.

4.5.3.4.1. Using Scratch Tapes

The reel field is omitted. The tape is returned to the scratch rack following use.

4.5.3.4.2. Saving Tapes

The reel field is omitted. A @SAVE statement is included which directs the operator to label the tape, return the reel number to the user, and file the tape in the saved tape rack following use. The user should contact the Director, RECC, or his Departmental Computer Coordinator, for information regarding service charges, etc.

4.5.3.4.3. Using Tapes Previously Saved

The reel number returned by @SAVE in the run creating the tape file is specified in the reel field. An 'N' or 'R' must be suffixed to the reel number. An 'N' directs the operator to mount the tape with no ring. If the program subsequently attempts to write on the tape, it will be terminated. An 'R' directs the operator to mount the tape ring in. The program may then either purposefully or inadvertently write over the information on the tape.

4.5.3.5. Examples of the Magnetic Tape @ASG Statement

The following are examples of the use of the @ASG control statement for tape files.

Scratch Tapes:

```
@ASG,T FILEY,T///36
```

File 'FILEY' is a temporary file requiring one unit of the system's choosing, and one or more scratch reels will be used. The noise constant is to be set to 36 characters.

```
@ASG,T 10,T
```

File '10' is a temporary file requiring one unit of the system's choosing, and one or more scratch reels will be used. A FORTRAN program may access this file by a statement of the following form: WRITE(10) A,B,C

Note: Whenever possible, Fastrand or word-addressable drum should be used for scratch files.

```
@ASG,TEM FILEX,T
```

File 'FILEX' is to be recorded in even parity and medium (556) density.

```
@ASG,T TEMP,T
```

File 'TEMP' is to be recorded in odd parity and high (800) density.

Tapes to be saved:

Any of the tapes assigned in the above examples may be saved by including a @SAVE statement in the runstream somewhere following the @ASG. For example, the tape file FILEX in the third example could be saved by including the statement

```
@SAVE FILEX
```

The reel number will be printed following the @SAVE statement and at the end of the print file.

Using tapes saved by previous runs:

```
@ASG,T FILEZ,T,U199N
```

File 'FILEZ' was previously created and saved for this user. The reel number is U199 and he does not want a write ring in the tape for this run.

```
@ASG,TEL TAPE,T,U180N/U50R
```

File 'TAPE' was previously created using even parity and low density (200). 'TAPE' requires one tape unit on any channel. Reels U180 and U50 are to be used with a write ring in U50.

4.5.4. The @MODE Statement

The @MODE statement is used to change the "mode" setting of a tape file. These modes are set initially when the @ASG statement is processed and may also be changed internally by use of the "set mode" function of the magnetic tape handler. The format of the @MODE Statement is:

```
@MODE,OPTIONS FILENAME,NOISE
```

The field 'FILENAME' is the same as for the @ASG statement. The file must be currently assigned to the run (an @ASG statement with this name must precede the @MODE statement). If the file is not assigned (never assigned or released via a @FREE statement), the run is placed in the error mode. The 'NOISE' subfield is optional. When specified, it is the same in form and meaning as for the @ASG statement.

The 'OPTIONS' field may contain the following options:

L - Low density
M - Medium density
H - High density
E - Even parity
O - Odd parity
I - Decimal (translate)
B - Binary (no translate)

With the @MODE statement, options (modes) are never assumed in the absence of others.

Diagnostic Messages

The generalized format as described in the section for the @ASG statement is used. The following table defines the meaning of the bits (1 = set) when set. Bits are numbered 35-0 reading left to right.

BIT	ACTION	DESCRIPTION OF MEANING IF BIT SET
35	K	Request not accepted - examine rest of bits as to why.
34	K	Field Error - noise constant
33	K	File has not been assigned to this run.
32		Unused.
31	K	Equipment type not tape for assigned file.
30		Unused.
'		
'		
18		

BIT	ACTION	DESCRIPTION OF MEANING IF BIT SET
17	K	Option conflict, only one option in the following 3 groups allowed (1) 'MLH' (2) 'EO' (3) 'BI'
16		Unused.
'		
'		
'		
0		

Note: K in ACTION field means that run will be terminated, unless demand.

4.5.5. The @CAT Statement

Cataloguing is normally done in the course of creating the file where the @ASG statement specifies that the file is to be catalogued. In this case, cataloguing is done when the run terminates or when a @FREE statement is found. It may be convenient to be able to catalogue one or more files without having them (and the required facilities) assigned to the run. The @CAT statement is used for this purpose. The file is catalogued but is not assigned to the run. No facilities are assigned. In any case, use of the @CAT statement is illegal if the named file is currently assigned to the run.

The format of the @CAT statement is identical to that of the mass storage @ASG statement, namely:

@CAT,OPTIONS FILENAME,TYPE/RESERVE/GRANULE/MAXIMUM

The specifications fields are interpreted as they are for the @ASG statement. However, the actual 'RESERVE' is not made. Allowable options are:

- R - Place in "read-only" state.
- W - Place in "write-only" state.
- P - Specifies that the file is to be catalogued as a "public" file rather than a "private" file.
- G - Inhibit computer center backup.
- V - Inhibit unload.

(See the mass storage @ASG section for a more detailed description of these options.)

'TYPE' should always be F2.

Examples of the @CAT Statement:

```
@CAT FILEX/A2962,F2
```

FILEX is to be catalogued on Fastrand II with the write-key A2962. The file is to be private; i.e., only the user cataloguing the file may assign it.

```
@CAT,P  PF,F2
```

The file PF is to be catalogued on Fastrand II. It is to be public; i.e., a run under any user-name may reference the file. Initial reserve is 0 tracks; granularity is TRK (track); maximum file size is 128 tracks.

Diagnostic Messages

(See @ASG Statement Diagnostics, section 4.5.2.6.)

BIT	ACTION	DESCRIPTION OF MEANING IF BIT SET
35	K	Request not accepted - examine rest of bits as to why.
34	K	Field error - illegal equipment type, etc.
33	K	File already assigned to this run.
32	K	Name already exists in directory of catalogued files.
31		Unused.
'		
'		
'		
18		Unused
17	K	Option conflict - only one option allowed from following groups (1) 'MLH' (2) 'EO' (3) 'BI' (4) 'RW'
16		Unused.
'		
'		
'		
0		

Note: K in ACTION field means that run will be terminated, unless demand.

4.5.6. The @FREE Statement

The @FREE control statement makes provision for the de-assigning of a file and the release of its input/output facilities. In the absence of a @FREE statement, the file and its facilities are held until run termination. Files should be de-assigned at the moment they are no longer needed so as to allow facilities, reels, and "exclusive use" areas to be assigned to other runs. The format of the @FREE statement is:

```
@FREE,OPTIONS FILENAME
```

where 'FILENAME' is either an external or an attached name (see the @USE statement). A warning diagnostic is given if the file has not been previously assigned. The 'OPTIONS' field may contain any of the following options.

- R - Releases the file assigned but retains the @USE name relationships to the filename and F-cycle.
- A - Releases only the @USE name relationship to the filename.
- B - Releases only the @USE name association to the filename if the attached name is not the only attachment. Otherwise, it acts like the blank option on the @FREE card freeing the file.
- C - Releases the file and all names associated with the file. (Same as no options.)
- D - Drops a catalogued file regardless of how it was assigned.
- I - Inhibits final cataloguing action if the file was assigned with a 'C' or 'U' option.
- X - Releases the exclusive use option set on the file but does not free the file.

A file that is named on a @FREE statement can no longer be referenced by the run; it can of course be reestablished by an @ASG statement provided its facility requirements can be met.

The actions taken by the system when a file is named on a @FREE statement are discussed below.

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

- MASS STORAGE - the mass storage area is made available as a file space for other runs.
- TAPE - Units are released for use by other runs. The currently mounted reel is rewound with interlock, indicating that the operator may remove it.

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

MASS STORAGE - Entry is made in the master directory and mass storage area containing the file is held. The file can now be referenced by other runs.

For a file being decatalogued (D or K option on @ASG):

MASS STORAGE --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.

A typical @FREE statement is shown in the following example of a partial control stream:

```
@ASG,C  FILEX,F2/3
@ASG,T  FILEY,T
.......
.......
@FREE FILEX
@FREE FILEY
```

FILEX is a Fastrand file to be catalogued and requires 3 tracks initially. FILEY is a temporary tape file requiring 1 tape unit. When the @FREE statement is encountered, FILEX is catalogued with the file area held for future reference. For FILEY, the tape is rewound with interlock, the unit is made available to other runs, and the operator will remove the reel and follow the user's instructions as to its disposal.

Diagnostic Messages:

The generalized format as described in the section for the @ASG statement is used. The following table defines the meaning of the bits (1 = set) when set. Bits are numbered 35-0 reading left to right

BIT	ACTION	DESCRIPTION OF MEANING IF SET
35	K	Request not accepted - examine rest of bits as to why.
34	K	Error other than syntax.
33		File is not currently assigned to this run.

32		Unused
.		
.		
.		
0		

Note: K in ACTION field means that run will be terminated, unless demand.

4.5.7. The @USE Statement

4.5.7.1. External, Internal, and Attached Names

The following classification of filenames is made.

External - The filename of the form given under Notation for filenames with which the file was initially catalogued and/or assigned.

Internal - A 1-to-12 character name by which the file may be referenced in an I/O request at the ER IO$ level. All I/O requests are made at this level but the user may not be directly aware of such requests since system software such as a processor or language library may make the request for him. The 'FILE' portion of the external name is automatically an internal name for the file, unless it duplicates another internal name.

Attached - A 1-to-12 character name by which the file may be referenced in an I/O request, other than the 'FILE' portion of the external name.

Note that attached names are also internal names. An attached name may be used on control statements; other internal names (i.e., the 'FILE' portions of external names) may not be used on control statements unless standard drop-out rules allow the omitting of all optional FILENAME subfields.

4.5.7.2. Format of the @USE Statement

The format of the @USE statement is as follows:

```
@USE  ATTACHED, EXTERNAL
```

or

```
@USE  ATTACHED, ATTACHED
```

where 'ATTACHED' is the 1-to-12 character internal name by which the file is referred to within programs or control statements following the @USE, and 'EXTERNAL' is the external name under which the file is assigned (and possibly catalogued) or to be assigned.

4.5.7.3. Use of the @USE Statement

The @USE control statement provides the user with the ability to refer to any particular file by two or more names. The need for the additional names arises from three conditions:

(1) Simplify run construction by allowing the equating of an external name to a shorter attached name.

(2) Resolve identical 'FILE' portions of external filenames.

(3) Connect names coded into programs to external or attached names.

The @USE statement allows the person setting up the run to choose external filenames descriptive to his run or to a particular catalogued file. It allows a particular internal name (in two or more executions) to point to different external files during the course of a run; or for different names to point to the same external file.

The @USE statement causes the 'ATTACHED' name to be linked to an external name. This external name may be directly specified on the @USE statement or may be linked to by the 'ATTACHED' name in specification 2. All such attached names are maintained for an external file. (The 'ATTACHED' name no longer points to any other external file, and if the file had a previous attachment, it is maintained rather than being deleted.) The list of attached names is always searched first on an I/O reference--with the 'FILE' portions of the external names used next on a no-find. If an 'ATTACHED' name is the same as the 'FILE' portion of some external file, that external file must have a @USE statement in effect before the file can be used. This may also be true for a recently assigned file (via @ASG statement), since the 'FILE' portion of its external name may be in the attached list, pointing to some other file. However, in this case the conflict can be removed if the reference is made by a control statement. The 'QUALIFIER' or at least the '*' will specify that the name is not an attached name.

4.5.7.4. Examples of the @USE Statement

Assume that the internal name 'FILEA' is the name in an I/O request, and the file 'PROJ1*FILEA' is assigned. Then 'PROJ1*FILEA' will automatically be used unless a @USE statement is presented making 'FILEA' point to different external file, for example, in the statement

```
@USE FILEA,PROJ1*FILEZ
```

'PROJ1*FILEZ' is then used for the I/O request.

4.5.7.5. File Name Uniqueness Within a Run

For each run, the executive maintains a table of all internal names assigned to the run. For each internal name, a pointer to the detailed description of the file is maintained. When an I/O request is made at ER IO$ level, only an internal name is given. In order to prevent ambiguity, this

internal name must be unique. Ambiguity would arise, for example, if the files A*LIB and B*LIB were concurrently assigned; or if the files ALPHA(1) and ALPHA(2) were concurrently assigned. In order to resolve ambiguity, a @USE must be done. If @USE INPUT, A*LIB and @USE OUTPUT, B*LIB were done, and I/O requests referenced the internal names INPUT and OUTPUT rather than LIB, there would be no ambiguity.

For convenience, the exec and system processors always do a dynamic @USE on files they are about to reference. Thus, if the statement @COPY A*LIB., B*LIB. (which calls the FURPUR processor) was given, FURPUR would automatically attach unique internal names to A*LIB and B*LIB. The processors always remove such name attachments via a @FREE,A statement before terminating.

Users should be wary of @USE statements such as @USE LIB,OTHER*LIB. The executive will issue a FACILITY WARNING message, since LIB will appear twice in the executive's table of internal names for the run. There is no actual ambiguity since both point to the same detailed file description; however, some processors detect the condition and hence refuse to process the file. It is best to keep names unique, say by @USE OLIB,OTHER*LIB.

4.5.8. The @QUAL Statement

The @QUAL statement allows the user to specify a filename qualification for implied usage on succeeding control statements involving filenames. The format of this statement is

```
@QUAL       QUALIFIER
```

where 'QUALIFIER' is a sequence of 12 or fewer characters used to qualify subsequent filenames which are headed by an asterisk (*). The 'QUALIFIER' is limited to the character set A...Z, 0...9, -, and $. An example of the use of the @QUAL statement follows:

```
@QUAL       JIM
   .
   .
   .
```

The subsequent statement

```
@FOR  *FILEA.JOE/ABC
```

would be interpreted as

```
@FOR  JIM*FILEA.JOE/ABC
```

Any number of @QUAL statements may appear throughout the control stream. Each will override the effect of the previous one.

4.6. Processor Call Statements

4.6.1. Notation for Program File Elements

A consistent notation is used throughout the system to reference elements of a program file. Using the COBOL syntax description notation, a reference to an element has the form:

```
[[FILENAME].]ELEMENT[/VERSION][(ELEMENT-CYCLE)]
```

'FILENAME' is described in sections 4.2.2.4 and 4.5.1.4.

An extensive series of dropout rules usually allow abbreviation of references to program file elements from the full form shown to something quite manageable.

The omission of 'QUALIFIER' with the '*' present causes the @QUAL statement to supply the qualifier used. If the @QUAL statement has not occurred, the user-name field from the @RUN statement is used as the qualifier. The omission of both the 'QUALIFIER' and the '*' causes the user-name field from the @RUN statement to be used as the qualifier, provided the 'FILE', if specified, is not an attached name which points to a particular filename. If the 'FILE' subfield is also omitted, then the run temporary program file, TPF$, is intended. The subfield 'ELEMENT' must always be present when referring to an element. The 'VERSION' subfield is required only in the case when more than one version of a particular element exists within the program file as is common when a program is in checkout.

When two or more specifications are used on a processor call statement, further abbreviation is possible if the 'FILENAME' part of a specification is identical to that part of the previous specification. In this case, only the period is needed before 'ELEMENT' to cause the previous 'FILENAME' to be assumed for this specification.

An 'F-CYCLE' number may be part of the 'FILENAME' field shown above. Its use is similar to that of the 'CYCLE' field discussed below and is described in the section on Notation for Filenames (4.5.1.4). Likewise, the two keys may be attached to the 'FILE' field. Their use and description are described in the section on @ASG Statements (4.5.2.2.1).

Note: On the various control statements, such as @ADD or @START, which can specify either a 'FILE' or an 'ELEMENT' name, a method is established which distinguishes between them. A period following the name will specify a 'FILE' and no period will specify an 'ELEMENT' in TPF$.

The cycle number serves to differentiate successive updates of a symbolic element. Omission of the cycle number when referring to a symbolic element implies that the most recently constructed copy is intended. A compacting method, as described later, is employed to prevent the retention of several cycles of a symbolic element from appropriating an excessive amount of space on whatever storage medium is employed. Some examples will help make this a bit clearer.

SORT	The element SORT in the run temporary file TPF$.
COST*PROG.EDIT	The element EDIT in the file COST*PROG.
*BACKUP.TLU/TWO	Version TWO of element TLU in file BACKUP. The qualifier for BACKUP is taken from the @QUAL control card.
PCF6.INTL(14)	The 14th generation of the element INTL in the file PCF6 belonging to the current username.

The notation given here for program file elements does not provide complete identification of the particular data desired since an element can exist in more than one form; for example, source language and relocatable. This is only an apparent ambiguity, however, since in all instances, the system is aware of the type of element desired.

4.6.2. Statement Format for Language Processors

There are several processors which process a source language element to produce a relocatable binary element. The general format of the statement for calling these processors is as follows:

```
@PROCESSOR,OPTIONS SI,RO,SO
```

The field 'PROCESSOR' may contain an acronym, including FOR, COB, ALG (FORTRAN, COBOL, ALGOL, respectively), in which case the indicated processor is called.

The field 'OPTIONS' may contain any one or several of the alphabetic characters 'A' through 'Z'. The use of any of these characters by a processor is defined in the pertinent processor users manual. However, the following have a common definition for all language processors and several system processors.

STANDARD PROCESSOR OPTIONS

A - Accept the results of processing even if errors are detected. In any case, do not error exit.

I - Initial insertion of a new source language input element from the control stream. The source language output ('SO') parameter is never used, as the source language input ('SI') parameter specifies the element name to be given to the source language output.

L - Produce the most comprehensive print listing available for this processor.

N - Produce the most abbreviated print listing available for this processor.

P - Specifies that source language output should be in Fieldata code. Identifies card image input, if any, as being Fieldata. (Compare with Q option.)

Q - Specifies that source language output should be in ASCII code. Identifies card image input, if any, as being in ASCII. (If neither P nor Q is specified, the code type of the existing source language input element, if any, is used. Otherwise, Fieldata is assumed.)

S - Produce a moderately comprehensive print listing.

U - Update an existing source language input ('SI') element to the next higher element cycle, thus saving any source language corrections that are currently being applied to the source language input element.

W - List correction lines at the head of the printer listing. (This is feasible only for a two-pass processor.)

X - Take error exit if errors are detected to inhibit further processing of the run.

The field 'SI' specifies the particular program file element to be used for the source language to be processed, in standard ELTNAME notation.

If present, and there is no I option, the lines immediately following the control statement are taken to be corrections to the source language element. If an 'I' option is present, then the lines following the control statement are given to the processor and are inserted into the program file as well.

The field 'RO' is the name of the element which is the relocatable or absolute code produced by a processor, in standard ELTNAME notation. This name (and the names associated with 'SI' and 'SO' fields) may include program filenames, F-cycle, keys, and version if desired. The name is not required because the name in the first specification field will be used if the field is blank.

The field 'SO' is the name of the source-language element produced by correcting the input source language element. If this field is void, no updated source language element will be produced unless a 'U' option is specified. In that case, an updated element is produced, with the same name and version as the input element, but with a cycle number one greater. No 'SO' field may exist when the I option is used.

For the three most common cases the specifications reduce to trivialities. If source language is coming from the control stream and no reference is made to program files on Fastrand, the processor call statement (assuming FORTRAN as an example) will reduce to

```
@FOR
.............
SOURCE LANGUAGE IMAGES
.............
```

In this case, the source-language program is compiled and the resulting relocatable element put into the run-temporary file, ready to be accessed by the COLLECTOR. The source language is not filed, but discarded following compilation.

A processor may be used to introduce a source-language element into a program file for the first time from the control stream. In this case, the I option is specified and there is no 'SO' field. As an example, consider the initial processing of the element WINDUP to be inserted into program file PF3.

```
@FOR,I PF3.WINDUP
.............
SOURCE LANGUAGE IMAGES
.............
```

In this case program file PF3 would be left with the source language and relocatable images of element WINDUP.

If an update is being made to some element, say WINDUP, in a program file (PF3), then the processor call statement would read:

```
@FOR,U PF3.WINDUP
.............
SOURCE LANGUAGE CORRECTIONS
.............
```

In this case, the source-language element specified by PF3.WINDUP is updated by the given correction lines and compiled.

The resulting relocatable element is inserted back into program file PF3, along with the next cycle of the source-language element WINDUP. If the input source-language element had a cycle number of, say 12, the new source-language element has a cycle number of 13. The entire element and correction lines are written in the program file, and the old element is marked deleted. If, for example, three cycles of source-language elements are being kept, the program file PF3 will contain, before the above statement is executed, the information:

```
WINDUP 10                          Complete Element
Correction lines converting WINDUP 10 to WINDUP 11
Correction lines converting WINDUP 11 to WINDUP 12
```

After compiling, PF3 contains:

```
WINDUP 11                          Complete Element
Correction lines converting WINDUP 11 to WINDUP 12
Correction lines converting WINDUP 12 to WINDUP 13
```

The number of cycles retained, say N, is a system standard set at system generation time; thus, a complete element and the N-1 most recent sets of corrections are kept. Normally this will involve considerably less mass storage space than even two complete elements and provides considerably more flexibility in backing up to some particular point in the history of a program.

The number of cycles kept is set at the system standard, unless some different number is specified by the program file utility routine for the particular element or particular file. The maximum number of cycles that can be retained is limited only by the storage space available, although the process becomes inefficient for an excessive number of cycles. It is possible to reference any particular available cycle of a source-language element. Suppose that cycles 10 through 12 of WINDUP are available. The processor call statement:

```
@FOR,U PF3.WINDUP(10)
```

would create a new cycle 11 and would delete cycle 12. On the other hand, the processor call statement:

```
@FOR PF3.WINDUP(11),.WINDUP,.WINDUP/NEW
```

would leave cycle 12 of WINDUP intact but would produce an entirely new source-language element WINDUP/NEW which would have a cycle number of 0. If there were any other cycles of WINDUP/NEW, they would be deleted, regardless of their cycle number.

4.6.3. Format of Correction Lines

Each processor optionally lists the source language input on which it is operating. On this listing, successive lines are labeled by successive integral numbers. When altering a source-language element in a program file, these numbers are used to indicate where corrections are to be inserted. A line of the form

```
-N,M
```

with the '-' on the first column indicates that source lines 'N' through 'M' are to be replaced by all succeeding lines in the control stream up to the next line with a '-' in column one, or the next control statement.

A line of the form

```
-K
```

indicates that succeeding corrections are to be inserted into the source language element following line K.

For example, the control stream

```
@FOR,U WEEKLY.REPORT
-30,31
CORRECTION LINE A
-100,115
-120
CORRECTION LINE B
CORRECTION LINE C
CORRECTION LINE D
```

will replace lines 30 and 31 by the correction line A, delete lines 100 through 115, and insert correction lines B, C and D following line 120.

If the user wishes to insert corrections before the first line item of his old source input, he must place them immediately after the processor call statement without specifying a correction line.

When corrections follow a processor call statement in a control stream, the source input routine (SIR) interprets a minus sign '-' in the first column of a line as a correction line. In certain situations where the user may have data with the '-' in column one, this is not desirable. This might happen when making corrections to a @RUN or @ADD stream with the

@DATA or @ELT processors. The user may wish to insert a set of corrections that are actually corrections for a processor call in the @RUN or @ADD stream. These corrections are not to be interpreted until the @RUN or @ADD is processed. To get around this problem SIR is prepared to handle the following correction line:

```
-=X
```

which says, from here on, SIR is to use 'X' to identify correction lines. 'X' may be 1,2, or 3 characters in length but must not contain a space or numeric character. The user may change correction line identifiers as often as he wishes but SIR will recognize only one identifier at a time. Initially SIR is set to recognize '-' as the correction line identifier.

The following example illustrates the use of identifier changes.

```
@DATA          FILE1,FILE2
-2                        Follow line 2
CORRECTIONS               with corrections.
-=*                       Change identifier to *.
*11,13                    Delete lines 11,12, and 13
CORRECTIONS               and insert corrections.
*=+++                     Change identifier to +++.
+++22                     Follow line 22
CORRECTIONS               with corrections.
@END
```

4.6.4. The System Program Files, SYS$*RLIB$, SYS$*LIB$, and TPF$

Relocatable library (SYS$*RLIB$). This file contains relocatable elements and procedure elements as needed by the system processors (compilers, collector, etc.). The file exists primarily as a place for standard relocatables to be used by the COLLECTOR in putting together programs and as a place for standard procedures to be picked up by the compilers.

System library (SYS$*LIB$). This file contains absolute elements only. This includes system processors like the COLLECTOR, FORTRAN, ALGOL, etc.

Additionally, there is a table in core containing the name, relative address in LIB$, and program size of selected processors. This table, LIBT, is referenced before LIB$ is referenced to save mass storage accesses.

Temporary program file (USER-NAME*TPF$). This file is created automatically by the executive when a run is initiated. The user does not have to specify the filename since a void filename can be used to reference the file. The file qualifier is taken from the user name field of the @RUN statement.

If a filename is not given, the order in which the executive searches program files is as follows:

On Processor Call Statement - LIB$, then TPF$
On @XQT Statement - TPF$

If a file other than TPF$ or LIB$ is intended on a processor call statement or TPF$ on an @XQT statement, the filename must be given. An example would be:

```
@XQT   FILEB.PROGA
```

where the program 'PROGA' is being executed from FILEB.

The search used by the compilers in finding procedures varies, but includes:

The file from which the symbolic element was taken,
and then SYS$*RLIB$.

See the appropriate compiler manual for additional files that may be searched.

4.7. Program Execution Statements

The program execution statements are used to control the construction, running, and diagnosis of a program created by a user.

4.7.1. The @MAP Statement

The @MAP statement is used for calling the COLLECTOR to collect a series of relocatable programs from one or more program files and to combine them into an executable program. The format of this statement is:

```
@MAP,OPTIONS  SI,RO,SO
```

The 'OPTIONS' field is essentially the same as for a language processor call statement. The fields SI, RO, and SO are used to specify the program file elements, if any, that contain or are to contain MAP directives, and the name of the output absolute element. A detailed description of the @MAP statement and examples of its use are included in the section on the COLLECTOR.

4.7.2. The @XQT Statement

The @XQT statement is used to initiate the execution of an absolute program prepared by the COLLECTOR. It has the format:

```
@XQT,OPTIONS              ELTNAME
```

The options subfield makes a 26-bit mask (each bit that is set represents an alphabetic character that was specified. A is represented by the 26th bit from the right, and Z by the rightmost). The 'ELTNAME' field of the statement names the program file element to be executed. Variations of the use of the @XQT statement are given in the section on the COLLECTOR.

Data cards to be input by the program may follow the @XQT statement. The program uses the system reference 'READ$' in gaining access to all images prior to the next executive control statement. When an executive control statement (other than an @EOF, see below) is detected by READ$, further reading by the user (or processor) is inhibited and an end-of-data return is given. Those images not requested by the program are bypassed when the program is finished. (A message denoting this is placed in run print file.) An example of the use of the @XQT statement would be

```
@XQT,BA   FILE1.PAYDAY
.......
USER DATA IMAGES
.......
ENDED BY NEXT CONTROL STATEMENT
```

where the options for controlling the program are 'B' and 'A', the file 'QUALIFIER' is taken as the 'user-name'. The 'FILE' portion of the filename is 'FILE1', and the element to be executed is 'PAYDAY', taken from 'FILE1'. If the element to be executed is in the run-temporary file, the filename is not needed. If such were the case, and options were not required, then the above @XQT statement would reduce to

```
@XQT PAYDAY
```

Additional examples of the @XQT statement are given in the section on the COLLECTOR.

4.7.3. The @EOF Statement

The @EOF statement is used as a file divider (general sentinel) within the data stream which follows the @XQT statement (or processor call statement). It is the only control statement that can be bypassed (read) by a user program. The format of the @EOF statement is

```
@EOF  S
```

where 'S' is a one-character sentinel, placed in column 6, to be passed to the requesting program at the time the statement is requested. When the @EOF is detected by READ$, an abnormal return is made to the requestor with the character found at 'S' made available. A subsequent request will cause the next image to be transmitted. An @EOF is never transmitted as such.

An example where the @EOF statement is used is

```
@XQT  PROGX
......
DATA OF PART 1
......
@EOF A
......
DATA OF PART 2
......
@XQT  PROGY
```

All cards between the two @XQT statements are to be input by PROGX. The @EOF statement serves as a marker between the two files.

4.7.4. The @PMD Statement

The @PMD statement may be used to obtain a post-mortem dump of all or part of the core storage used by an execution task. The format of this statement may take one of two forms:

```
@PMD,OPTIONS  NAME1,NAME2,NAME3,...,NAMEN
```

or

```
@PMD,OPTIONS NAME,START,LENGTH,FORMAT
```

Detailed discussions of each of these forms are given in the section on DIAGNOSTIC AIDS. Hence only a brief summary is given at this point. All @PMD statements following an execution are honored until a control statement is encountered which is not a conditional control statement (i.e., @SETC, @TEST or @JUMP) or a @EOF statement. (Any other statement will cause the termination of the PMD mode.) The available 'OPTIONS' are divided into two classes--special and standard. If a special option is used, the first form of the @PMD statement is required, and 'NAME1', 'NAME2', etc., are names of segments or elements which are to be dumped according to the 'OPTIONS' specified. If only standard options are specified, the second form is used. The field 'NAME' may specify an element or segment to be dumped, or it may be void, in which case all of the user's area of core is dumped. If 'NAME' is specified, 'START' and 'LENGTH' specify an area of the element or segment to be dumped, and 'FORMAT' specifies a format to be used for the dump listing.

Standard options allow for conditional dumps, depending upon the termination of the run, for changed-word dumping and for dumping all of blank common. Special options allow for dumping all of an element or segment; or only bank 1 or bank 2 portions of an element; and for specifying only elements which are not to be dumped.

4.8. Conditional Statements

4.8.1. Purpose of Conditional Statements

The conditional statements are set apart from other executive control statements because they are special-use features and need not be of concern in many applications.

The conditional control statements are used to accomplish dynamic adjustment of the control stream as it is being executed. A common "condition" word is maintained by the system throughout the course of a run. The value in the "condition" word is referenced (tested or set) from within the control stream via the conditional statements, causing portions of the stream to be bypassed. In addition, all user programs within the run and the executive have the ability to access the word and/or reset their respective thirds (see the section on the "condition" word). This method may cause the user program to take different paths and/or to set parts of the word such that portions of the control stream are skipped. This conditional network

allows a given control stream to produce many different results with only a slight modification to the stream or with no modification if the effective stream is dictated by user programs reacting to stimuli such as amount of data, data, day of month, time of day, etc.

4.8.2. Statement Labels

The executive language is such that control statements may be labeled. This feature is provided in order to allow functions (statements) to be skipped with control being passed to a statement with a particular label. The @JUMP control statement (described later) is used to move control to a statement with a particular label.

As described in the first part of this chapter, the labeled executive control statement has the format:

```
@LABEL:COMMAND,OPTIONS  SPEC1,SPEC2,...,SPECN  COMMENT
```

where the label is limited to six characters from the alphanumeric set (A...Z,0...9), begins with an alphabetic, and is immediately followed by the colon (:).

An example of an @XQT statement that is labeled is

```
@A:XQT  PROGX
```

where 'A' is the label and 'PROGX' is the element to be executed.

A label specification on certain control statements is meaningless and will be ignored. Those statements which fall into this class are @RUN, @EOF, @COL, @ BIN, @PWRD, and @END.

A label (or labels) may be attached to an existing control statement without physically changing the statement, by use of the @LABEL statement (see below).

4.8.3. The @LABEL Statement

A label can be placed on an existing control statement by placing a @LABEL statement immediately preceding the existing statement. The format of the statement is

```
@LABEL:
```

where 'LABEL' is the tag to be attached. If a label is also present on the existing statement, the statement is recognized by both labels. If more

than one @LABEL statement is present, all are attached. As an example, the @XQT statement below can be referenced by both the label 'A' and the label 'B'.

```
@A:
@B:XQT   PROGX
```

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

4.8.4. The "CONDITION" Word

The system maintains a "condition" word (computer word of 36 bits) for each active run. The "condition" word is set to zero at the beginning of a run (in the absence of a 'SET' specification on the @START statement). This word is divided into three parts from left to right. The left third may be set by the executive only (for error conditions, etc.). The middle third may be set externally in the control stream via @SETC, and the right third is set by the internal user program via an internal reference to SETC$. User programs can retrieve the entire word (via an internal reference to COND$) and the word can be tested from within the control stream, causing branching to a particular statement, via the @TEST control statement. A @JUMP statement is provided for branching when a particular test is met.

The state of the "condition" word, whether set from the control stream or by user programs, can be monitored at any point within a run to decide how the run should best proceed.

The executive uses the left third of the condition word to indicate the type of program termination. The values that may be found in this portion of the condition word, and their corresponding meanings, are given below:

0. EXIT$ - termination of all activities.

1. EXIT$ - termination of last activity; ERR$ - termination of one or more previous activities.

2. ERR$ - termination of last activity; EXIT$ - termination of all previous activities, if present.

3. ERR$ - termination of last activity; ERR$ - termination of one or more previous activities.

4. ABORT$ - termination of last activity; EXIT$ - termination of all previous activities, if present.

5. ABORT$ - termination of last activity; ERR$ - termination of one or more previous activities.

A value of 2 or 3 causes a batch run to be terminated after processing PMD control statements and conditional statements. A value of 4 or 5 will cause immediate termination of a batch run.

4.8.5. The @SETC Statement

The @SETC control statement is used to store (set) a value in the second third of the "condition" word. The format of the statement is

```
@SETC,OPTIONS  VALUE/J
```

where 'VALUE' must be specified and 'J' is optional but assumed to be T2 if absent. The 'VALUE' subfield contains a positive, octal number not to exceed 4 digits. It is treated as 36 bits (right justified, zero filled) prior to the partial-word store in the "condition" word. If the magnitude of the number is greater than can be contained in the 'J' designated portion of the "condition" word, truncation occurs.

Allowable 'J' designators are

T2	Middle Third
S3	Third Sixth from Left
S4	Fourth Sixth from Left

Examples of the @SETC statement are

```
@SETC 6
```

where the second third of the "condition" word is set to 6 or 0006, and

```
@SETC  10/S3
```

where the third sixth is set to 10 octal, with the rest of the word left undisturbed.

Options allowed are I and A.

- I - set indicator in T1 of the run condition word to inhibit termination of a batch run following error termination of a program or processor. This does not inhibit run termination due to abort terminations.
- A - turn off the "inhibit termination" indicator set by the I option.

4.8.6. The @JUMP Statement

The @JUMP control statement is used when statement execution is to be branched to a particular labeled statement. The format of the statement is

```
@JUMP  LABEL
```

where 'LABEL' appears as a label on a subsequent control statement, or is a decimal numeric (N) specifying that control is to be passed to the Nth control statement that follows, except that those statements which cannot have labels are not considered in the count. Note that the @JUMP statement must reference in the forward direction (to a statement not yet processed). A numeric of zero (0) is illegal.

4.8.7. The @TEST Statement

The @TEST control statement is used to test the value of the "condition" word for the purpose of selecting particular control statements to be executed (or skipped). The format of the @TEST statement is

```
@TEST  F/VALUE/J,F/VALUE/J,...
```

where 'VALUE' contains a positive, octal number not exceeding 12 digits. The 'J' field is optional and when it is not specified the middle third is assumed. Allowable 'J' designators are

W	Whole Word
H1	Left Half
H2	Right Half
T1 thru T3	Left Third thru Right Third
S1 thru S6	Left Sixth three Right Sixth

The 'F' field (function field) specifies the test to be made. If more than one function appears on the statement, scanning continues until a test is met or all functions are exhausted. The control statement immediately following the @TEST statement will be skipped if a test is met; otherwise, it will be executed. Allowable functions are:

TE	Test Equal (Skip the next control statement if the 'J' designated portion of the "condition" word is equal to 'VALUE' or in simpler terms, skip if C equals V)
TNE	Test for not Equal (Skip if C not equal to V)

TG	Test for Greater (Skip if C greater than V)
TLE	Test for Less than or Equal (Skip if C less than or equal to V)

The specified 'VALUE' is interpreted in the same manner as for the @SETC statement (full 36 bits). However, it will appear negative if the uppermost bit is set. This is also true for the "condition" word when the entire word or a third is being tested.

Note: The @SETC statement is equivalent to the machine instruction 'STORE A'(SA) where the 36-bit 'VALUE' is found in the 'A' control register.

The test functions are equivalent to the machine instructions 'TE, TNE, TG, and TLE' where the 36-bit 'VALUE' is in the control register.

An example of the use of the @TEST control statement would be

```
@TEST TE/6/T2,TG/12/H2
@XQT  PROGX
```

If the middle third of the "condition" word is equal to 6, or if the right half is greater than 12, the @XQT statement would be skipped; otherwise, it would be executed.

Consider the following run which utilizes all three of the conditional control statements @SETC, @TEST and @JUMP:

```
@RUN        ID6,PROJ1,888294,10
@SETC       6                    Initial set of "condition" word
......
......
@TEST       TE/6
@XQT        PROGX
......
......
@TEST       TE/6
@JUMP       2
@JUMP       A
@TEST       TE/10,TE/4
@JUMP       3
```

```
@SETC       4
@JUMP       B
@TEST       TE/11
@JUMP       C
@XQT        PROGY
......
......
@A:XQT      PROGA
......
......
@B:XQT      PROGB
......
......
@C:XQT      PROGC
@FIN
```

By changing the value (now 6) on the initial @SETC statement, the run can be made to produce different results. As the run is now "set", the programs A, B, and C will be executed. If the initial "set" value were 3, then program X would also be executed. If octal 10 or 4, programs Y and A would be skipped. If 11, all programs are executed. If some other number, Programs Y, A, and B are skipped.

Although not shown in the example, it is important to note that PROGX, if executed, could have set some part of the right third of the "condition" word. In the example above, this would not have affected the paths taken, but if any part of this third were tested via @TEST, it would have had a part in determining whether the tests were met or not. The same is true concerning the executive third.

4.9. Statement Syntax Error Diagnostics

While the control statement interpreter is converting the control statements from external to internal format, it performs a syntax check on each statement. Below are the error messages that may occur on the printer, immediately following the statement, when a syntax error is detected.

1. XX Illegal Option Z
2. XX Illegal Character Z
3. XX Max Number of Characters Exceeded
4. XX Max Number of Fields or Subfields Exceeded
5. XX Required Field or Subfield Missing
6. @ in Column 1 of Continuation Card

where

XX = the character position at which the error was detected.

Z = the illegal character or option.

5. FILE UTILITY ROUTINES (FURPUR)

5.1. General

In addition to the executive control statements discussed, there is a set of statements recognized by the executive as calls for the file utility routines (FURPUR). When the executive encounters a FURPUR statement it loads the FURPUR processor. FURPUR continues by processing statements until the executive signals the next statement is not a FURPUR statement.

Statements processed by FURPUR are listed below by command name with a brief description of the functions they perform.

Command	Function
@CHG	Changes the name and/or version of program file element.
@CLOSE	Writes two hardware end of file marks on a tape and rewinds it.
@COPIN	Reads elements in element file format on tape and inserts them in a program file.
@COPOUT	Writes elements of program files to tape in element file format.
@COPY	Transfers files or program file elements from one file to another.
@CYCLE	Sets a new maximum number of cycles to be retained for a symbolic program file element.
@DELETE	Drops a file in the directory or marks a program file element deleted.
@ENABLE	Removes the disable flag from temporarily disabled catalogued files.
@ERS	Releases space allocated to program files.
@FIND	Locates an element on a tape in element file format and positions the tape before the element's label block.
@MARK	Writes a hardware end of file mark on tape.
@MOVE	Moves a tape forward or backward over a specified number of hardware end of file marks.
@PACK	Rewrites a program file to exclude elements marked deleted.
@PCH	Punches program file elements.
@PREP	Creates an entry point table for a program file.
@PRT	Lists directory items for catalogued files, the table of contents of a program file, or the text of a symbolic element.
@REWIND	Rewinds tapes.

5.2. Statement Format

The general form of a FURPUR statement is:

```
@LABEL:COMMAND,OPTIONS SPEC1,SPEC2,...,SPECN
```

5.2.1. Contents of Specification Fields

A specification field may contain a filename, a filename and element name, or a parameter value depending on the statement and its intended use. External or internal filenames may be used. The internal names $FILEA, $FILEB and the external filename MSDGET should not be used in runs containing FURPUR statements. External filenames take the form:

```
QUALIFIER*FILE(F-CYCLE)/READ-KEY/WRITE-KEY.
```

Element names take the form:

```
ELEMENT/VERSION(ELEMENT-CYCLE)
```

The filename should be followed by a period. If the specification requires an element name, it should follow the period. The element cycle or the F-cycle may be excluded when relative zero (0) is intended. System drop-out rules for the qualifier apply.

5.2.2. File Assignments

FURPUR will automatically attempt to assign catalogued files not assigned at the time the FURPUR statement is encountered. FURPUR will require exclusive use of the files named in many cases, and therefore will attach exclusive use as necessary to files assigned by the user. FURPUR returns the file to the assigned status it had prior to the statement, except when the function of the statement itself is to change the status (e.g., @DELETE). Temporary files must be assigned by the user.

5.2.3. Options Field

In general, the options used vary with the statement. The options below have the same meaning for all FURPUR statements.

C	Requests that FURPUR exit normally after an error condition. A diagnostic message will still be printed. Without the C option, an error condition will cause FURPUR to exit to ERR$. The C option may be used on any FURPUR statement.
A	Process absolute elements.
R	Process relocatable elements.
S	Process symbolic elements and procedure elements.

Additional options are discussed with the commands to which they apply.

5.3. Shorthand Notation

The filename may be omitted from the specification field on all FURPUR statements. If the filename is omitted in specification 1, TPF$ is substituted. If the filename used in specification N is the same as that in specification N-1, it may be omitted, provided the element name in specification N is preceded by a period.

The period may be omitted on any FURPUR statement not containing an asterisk (*) other than @CYCLE,@PRT,@COPIN (V option), and @COPOUT (V option) that does not specify A, R, or S options. If the filename TPF$ was omitted, the period may also be omitted on the @PCH statement and @PRT statement.

5.4. FURPUR Statements

The following paragraphs discuss in detail the statements processed by FURPUR.

5.4.1. @COPY

The COPY command is used to copy a file or element from one file to another.

5.4.1.1. Formatting the @COPY Statement

The @COPY statement has the following format:

```
@COPY,OPTIONS SPEC1,SPEC2,SPEC3
```

SPEC1 is the input file or element to be copied.

SPEC2 is the output file to be copied into.

SPEC3 is used only for tape to tape copying of entire files with no options or with the 'M' option. It specifies the number of input files to copy to the output tape. If SPEC3 is omitted, one file will be copied. The copy is terminated regardless of the value of SPEC3 if a void file is copied. The input tape will be left positioned following the end of file after the last file is copied. The number of blocks in each file copied, and the number of files copied will be indicated by messages.

The options allowed are:

(No option)
or 'M' only
or 'N' only
or 'MN' only

The 'M' option, when used, is valid only if the output file is on tape. It indicates the output tape is to be marked with a hardware end of file mark after each non-void file copied from the input tape. In addition, a second end of file is written following the last non-void file copied, and the tape is then backspaced one end of file mark.

If the input file is on tape, and the output file is on Fastrand, the blocks will be copied to Fastrand in a contiguous manner beginning in sector 0. Note that a block size not divisible by 28 will leave a garbage area in the last sector of the block as it appears on Fastrand.

If the input file is on Fastrand and the output is on tape, track size blocks will be written.

The 'N' option, when used, is valid only if the input file is on tape. It indicates that non-integral blocks (not containing a multiple of 6 characters) are to be zero-padded. In the absence of the 'N' option, the copy is terminated upon encountering a non-integral block.

V (and 'M')

A file is to be copied. SPEC1 names the input file and SPEC2 names the output file. The input file and output file may not both be on tape or on Fastrand. If the input file is on Fastrand, variable block size will be assumed. This means the first word of each block contains the block size. This word is stripped before the block is written to tape. The 'M' option is valid only when the output file is on tape. It indicates that after the file is copied, a hardware end of file mark is to be written on the output tape. This is done by writing two end of file marks and backspacing one.

If the input file is on tape, a word containing the block size will be prefixed to the block before it is written on Fastrand.

Note that a @COPY to or from Fastrand begins in sector 0. Each block on Fastrand starts in a new sector. A garbage area exists in the last sector of blocks whose size is not divisible by 28. The input file must be followed by a hardware end of file if on tape.

G (and 'M')

A file to be copied. SPEC1 names the input file and SPEC2 names the output file. The input and output files should not both be on Fastrand or both on tape. The 'M' option is valid only if the output file is on tape. It indicates that after the file is copied two hardware end of file marks are to be written, and the tape backspaced one.

If the input file is on Fastrand, each allocated track beginning with relative track zero will be prefixed with its relative track number and then written to tape. In this case, the @COPY is terminated by an attempt to read outside the file limits. The first block written on tape will be a label block that indicates the format.

If the input file is on tape, the first word of each block indicates the relative track the block (minus the first word) is to be copied onto on Fastrand. The @COPY is terminated when a hardware end of file mark is encountered on the input tape.

This option supplies the user with an efficient means of saving and recreating a Fastrand file that contained voids. Such voids are created by random access techniques.

F — A file is to be copied. SPEC1 names the input file, and SPEC2 names the output file. The input file must be in system data format (SDF). Reading of the input file is terminated by an SDF end of file sentinel. If the output file is put on tape, two hardware end of file marks will be written and the tape backspaced one. Block sizes for files on tape must be 224 words. Program files or element files may not be copied with this option.

I — The I option adds a file in SDF format to a program file as an element. SPEC1 names an input file which must be SDF format. SPEC2 names a file in program file format on Fastrand and the element name to be entered in the program file element table. The new element will be entered as a symbolic element, cycle 0. The cycle limit is set to the system standard.

A,R,S — Selected elements from a file in program file format on Fastrand are to be reproduced in another file in program file format on Fastrand. SPEC1 names the input file, and SPEC2 names the output file; or, SPEC1 names the input file and input element, and SPEC2 names the output file and output element. Only non-deleted elements of the input file may be inserted in the output file. The options indicate the element types to be reproduced. If no element name is given, all elements of the types indicated will be reproduced in the output file. Any combination of A,R,S may be given.

P — All non-deleted elements of one file in program file format are to be inserted in another file in program file format. SPEC1 names the input file. SPEC2 names the output file.

When reproducing elements via the A,R,S, or P options, the related procedure name entries are added to the output file's procedure name table entries. If a relocatable element is reproduced the output file's entry point table will be destroyed. A @PREP statement can be used to recreate the entry point table when necessary.

5.4.1.2. Examples of the @COPY Statement

T, T1, T2,... will be used to indicate tape files; F, F1, F2... will be used to indicate Fastrand files.

Some typical @COPY statements are given below.

```
@COPY F1.,F2.
```

The Fastrand file F1 is copied into the Fastrand file named F2.

```
@COPY,M T1.,T2.,9
```

Nine files on tape with filename T1 are copied onto tape with filename T2. Each file on T2 is separated by end of file marks as directed by the 'M' option. The last file on T2 is followed by 2 end of file marks. T2 is left positioned between the last two file marks.

```
@COPY,GM F.,T.
```

File F is copied to tape with filename T, in @COPY,G format. The hardware end of file marks are written following the file. The tape is then backspaced one end of file mark. If F was a program file, sequential access of elements in the output file via @FIND and @COPIN are not permitted since they require element file format. Note that the entire file as it was before the copy, including all tables of contents and deleted elements, will be reproduced when the file is returned to Fastrand via the 'G' option. Two program files saved on tape via this option may not be merged as each file would overlay the other.

```
@COPY,P F1.,F2.
```

The non-deleted elements of program file F1 are reproduced in program file F2. F1 is unchanged.

```
@COPY,I F.,F1.ELT1/VERS
```

The file F in SDF is reproduced in program file F1. An entry is created in the element table of F1 with an element name ELT1, version name VERS, cycle 0, whose text area contains the contents of F.

```
@COPY,I T.,F1.ELT1/VERS
```

Same as previous example, except that the input file is on tape.

```
@COPY,RS F1.,F2.
```

The non-deleted relocatable elements and symbolic elements of program file F1 are reproduced in program file F2. F1 remains unchanged.

```
@COPY,RS F1.A,F2.B
```

The symbolic and relocatable elements with name A in program file F1 are added to program file F2 with the name B. Both types R and S must exist in F1 as non-deleted elements.

5.4.2. @COPOUT

COPOUT is used to write a program file, or selected elements of a program file to tape in element file format.

5.4.2.1. Formatting the @COPOUT Statement

The format of the @COPOUT statement is:

```
@COPOUT,OPTIONS SPEC1,SPEC2
```

The @COPOUT statement is used to write elements of program files on Fastrand onto tape in element file format. Procedure name entries will be preserved. Entry points are not preserved. Element file format was designed to reduce tape movement when it is necessary to read selected elements from tape, as opposed to treating a group of elements as a single file. Tapes must be in element file format in order to use @FIND or @COPIN.

The available options are:

No option	All non-deleted elements of a program file are written onto a tape in element file format. Two hardware end of file marks are then written on the tape and the tape is then backspaced over the second one. SPEC1 names a program file on Fastrand. SPEC2 contains a filename that refers to tape.
A,R,S	Non-deleted elements of the types specified by the options are written onto tape in element file file format. SPEC1 names a program file on Fastrand and SPEC2 contains a filename that refers to tape and the name to be given to the element written to tape. Any combination of options may be used whether applied to the entire file or to a single element name. If no element name is given in SPEC1,

all non-deleted elements of the types specified by the options are written to the tape with the filename given in SPEC2. If an element name is included in SPEC1, all types specified by the options for the element name given will be written to the tape with the element name given in SPEC2. Each type specified must refer to a non-deleted element.

V — Non-deleted elements of a program file selected by version name and type are written in element file format on tape. The V option may be used in combination with the A,R, and S options. The V option alone implies all element types are to be considered. SPEC1 names a program file and an element version name. SPEC2 contains the filename of a tape and element version name. The elements with the version name given in SPEC1 of the types specified by the options will be written to the tape indicated by SPEC2 in element file format. The version name given in SPEC2 replaces their original version name. If the version name in SPEC2 is omitted, the elements written to tape will have the same version name as in SPEC1.

5.4.2.2. Examples of the @COPOUT Statement

The @COPOUT statement is typically used in the following manner:

```
@COPOUT     PROGRAM.,HOLDPROG.
```

The program file named PROGRAM will be copied onto the output file HOLDPROG. It will be reformatted as an element file. The R,S, and A options apply as with the COPY statement.

```
@COPOUT     A,TAPE
```

In the above example the A in SPEC1 is presumed to be a filename and the entire file will be copied to TAPE. The file is marked with an end of file mark because no options are present.

```
@COPOUT,ARS     C.,D.
```

The contents of file C are copied to file D and no EOF mark is written.

```
@COPOUT,R     A.,B.
```

In the above case all relocatable elements are copied from file A to B.

```
@COPOUT,S     A.B,C.D
```

In the above case the 'S' option indicates to the processor that it is to handle one element, or type of element. That element will be copied to the file named in SPEC2 ('C') and given the element name named in SPEC field 2.

```
@COPOUT     A.B,C.
```

In this case the element name 'B' is ignored and the entire file A is copied to File C, because no option letters are present. Entry points will not be copied.

```
@COPOUT,SV     A./B,C.
```

All symbolic elements in file A with a version 'B' will be copied to File C.

```
@COPOUT,AV     A.,C.
```

All absolute elements in file A with no version name will be copied to File C.

5.4.3. @COPIN

The @COPIN statement is used to read elements from a tape in element file format and insert them in a program file on Fastrand. Related procedure names are entered in the program file's procedure name table. Entry points are not added to the entry point table. If a relocatable element is added to the program file as a result of the @COPIN the entry point table will be destroyed if one existed previous to the @COPIN. The @PREP statement may be used to recreate the entry point table when necessary. If a tape error occurs, only those elements entered properly prior to the error will appear in the program file's table of contents.

5.4.3.1. Formatting the @COPIN Statement

The format of the @COPIN statement is:

```
@COPIN,OPTIONS SPEC1,SPEC2
```

The allowable options are:

No option	SPEC1 contains the filename of a tape positioned at the label block of an element in element file format. SPEC2 names a program file on Fastrand. Elements are read from the tape named by SPEC1 and inserted in the program file named by SPEC2 until a hardware end of file mark is encountered.

A,R,S — SPEC1 contains the filename of a tape positioned at the label block of an element in element file format. SPEC2 names a program file on Fastrand, or SPEC1 contains the filename of a tape and the name of the element in element file format at whose label block the tape must be positioned, and SPEC2 contains the name of a program file on Fastrand and the element name to be given to the element when it is added to the file.

V — SPEC1 must contain the filename of a tape positioned at the label block of an element in element file format, and the version name of the elements to be added to the program file on Fastrand named by SPEC2. SPEC2 also contains the version name to be given the elements added. Elements not having the same version name as given in SPEC1 are skipped. The @COPIN is terminated when a hardware end of file is encountered. If only those elements with no version name are to be added, the version subfield may be omitted in SPEC1. If the version subfield is omitted in SPEC2, the elements will retain their same version name. The V option may be used in any combination with the A,R, and S options, if only selected types are to be added.

5.4.3.2. Examples of the @COPIN Statement

The @COPIN statement is typically used in the following manner:

```
@COPIN      HOLDPROG.,PROGRAM.
```

In this example, the element file HOLDPROG is copied and reformatted on the Fastrand area assigned to file PROGRAM. When the @COPIN operation is complete, file PROGRAM will be in the standard program file format and may be treated as a program file in any subsequent operation.

```
@COPIN,R      TEMP.ELTA,PF1.
```

The above command causes the relocatable element ELTA to be read from the element file TEMP and added to the program file PF1. The element file must be positioned at ELTA (e.g. with @FIND). The entry point table is not updated (this may be done with @PREP).

```
@COPIN,RV     A./B,C.
```

All relocatable elements in file A with a version 'B' will be copied to File C.

```
@COPIN,SV     A.,C.
```

All symbolic elements in file A with no version name will be copied to File C.

5.4.4. @DELETE

The @DELETE command may be used to drop a catalogued file or to mark an element of a program file on Fastrand deleted. The effect of the @DELETE command for catalogued files is the same as the sequence:

```
@ASG,A   FILENAME
@FREE,D  FILENAME
```

When used to mark a program file element deleted, the element entry and its related procedure names are marked deleted. The element may be removed subsequently by using a @PACK statement.

5.4.4.1. Formatting the @DELETE Statement

The @DELETE statement has the following format:

```
@DELETE,OPTIONS SPEC1,SPEC2,...,SPECN
```

The available options are:

(No options) — A catalogued file is to be dropped. Each specification field names a catalogued file. The filenames given may be external or internal.

If an external name is given, the F-cycle to be dropped should be specified. The latest cycle is understood if none is given. Security keys must be given if the file was catalogued with keys and the file is to be assigned by FURPUR. The keys will be ignored if the file is already assigned.

If an internal name is given the external name on the associated @USE statement must satisfy the same rules as if it had appeared on the statement itself. Note that when the file is actually dropped from the directory older F-cycles will have their relative F-cycle value increased one. The file is not actually dropped from the directory until all assignments made before the drop flag was set have been @FREE'd.

A,R,S — An element of a program file is to be deleted. Each specification field given names an element and the program file that contains it. The types given by the options will be marked deleted. Any combination of A, R, and S options may be used, but at least one must be given. @DELETE requires each element as specified by name and options on the statement exist in a non-deleted state prior to the @DELETE statement. Including a cycle number for symbolic elements is illegal on the @DELETE statement. Associated entry points and procedure names will also be marked deleted.

5.4.4.2. Examples of the @DELETE Statement

```
@DELETE,S F.ELT1/VERS,F1.ELTY
```

The symbolic element ELT1/VERS in program file F, and the symbolic ELTY in program file F1 will be marked deleted. Associated procedure names, if any, will also be marked deleted.

```
@DELETE F.,T1.
```

The catalogued filed F and T1 are dropped from the directory (only the most recent F-cycles).

5.4.5. @PRT

PRT is used to print the table of contents of a program file, or to print the master file directory items of catalogued files, and to print the text of symbolic elements.

5.4.5.1. Formatting the @PRT Statement

The @PRT statement has the following format:

```
@PRT,OPTIONS SPEC1,SPEC2,...,SPECN
```

The available options are:

T (and L)	List the table of contents of a program file on Fastrand, or the table of contents items for all elements with a given name and version. Each specification field names a program file (with a trailing period) or a program file and element name. See Notes on @PRT,T, the second section following. A compressed format is used for a demand run, unless the L option is used.
(no option)	If no specification fields are given, directory information for all files catalogued under the current USER-NAME will be listed. The directory items listed will not include security keys. Items are listed in alphabetical order, first by reference number, then by qualifier and filename. If at least one specification is given, a symbolic element of a program file will be listed with line numbers. Each specification field given should contain the name of a program file on Fastrand and the name of a symbolic element within the file. An element cycle may be specified. If none is specified, the latest cycle is listed.
F	List the directory item for a file catalogued under the current USER-NAME. Each specification field names a catalogued file. If the specified file was not catalogued by the current user, an error is indicated. Security keys are not listed.

N — List the directory items for all files catalogued with the current USER-NAME and the reference number specified. Security keys are not listed. Each specification field names a reference number. If no specification fields are given, action is identical to @PRT (no options).

5.4.5.2. Examples of the @PRT Statement

```
@PRT,T PROGFILE.
```

The table of contents of the program file PROGFILE, on Fastrand, is listed. The tables listed include the element table, procedure name tables, and the entry point table. Element table entries are listed in the order the elements were introduced into the file. Other tables are listed in alphabetical order.

```
@PRT PROGFILE.SAM/XYZ
```

The latest cycle of the element SAM, version XYZ in program file PROGFILE, is listed.

```
@PRT
```

A listing of the directory items for all files catalogued under the name on the @RUN statement is generated.

```
@PRT,N 11F196901,03B691910
```

A listing of the directory items for all files catalogued under the name on the @RUN statement and reference numbers 11F196901 and 03B691910 will be generated.

```
@PRT,F FILE1.
```

A listing of the directory item for FILE1 is generated, assuming that the name fields agree. If not, an error is indicated.

```
@PRT,T  PROGFILE.MAIN/ONE
```

A listing of the table of contents items for all elements with the name and version MAIN/ONE in program file PROGFILE is printed. These elements may include symbolic, relocatable, absolute, deleted, and non-deleted elements.

5.4.5.3. Notes on @PRT,T

PRT Format with T Option

The contents of the individual print lines are as follows:

Element Table (1 item per line)	No. of Characters
Delete Flag	1
Element Name	12
Version	12
Type	11
Date and Time	19
Element Sequence No.	6
Location (Rel. Sector #)	11
Length in Sectors (Including Text and Preamble)	6
Cycle Limit	4
Latest Cycle No.	4
No. of Cycles	4
	90 characters

Assembler and FORTRAN Procedure Tables (3 items per line)	No. of Characters
Delete Flag	1
Procedure Name	12
Location in File (Relative Word Number)	11
Element Link (Element Sequence Number)	6
	30 characters

COBOL Procedure Table (2 items per line)

Delete Flag	1
Procedure Name	30
Location in File (Relative Word Number)	11
Element Link (Element Sequence Number)	6
	48 characters

Entry Point Table (5 items per line)

Delete Flag	1
Entry Point Name	12
Element Link (Element Sequence Number)	6
	19 characters

Explanation of Title Headings

Element Table

D-Flag	An asterisk means entry deleted. No other symbol is used.
Type	If the element is symbolic, the processor which created it is indicated.
Date-Time	Time that element was added to this file.
Sequence Number	Position of the element in this file (this is sequentially issued) as elements are added to the file.
Text Size	This is the text size in sectors. A sector is 28 words.
Pre Size	For relocatable elements, the preamble length is given in sectors (28 words).
Cycle Word	The cycle word is broken up into three separate parameters; starting from left to right they are: 1) the number of cycles the system will maintain (maximum). 2) the number of the most current cycle (absolute). 3) the number of cycles currently being maintained.
Location	Refers to the sector position relating to the start of the file.

Procedure Table
Assembler, COBOL
FORTRAN

D-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,T Command from a Demand Terminal

a) The TOC format from demand stations is:

TYPE ELEMENT NAME/VERSION(CYCLES).

The 'TYPE' will be represented as follows:

ELT Symbolic Elements - ELT
ASM Symbolic Elements - ASM
COB Symbolic Elements - COB
FOR Symbolic Elements - FOR
ALG Symbolic Elements - ALG
MAP Symbolic Elements - MAP
DOC Symbolic Elements - DOC

Assembler Procedures	-	ASMP
COBOL Procedures	-	COBP
FORTRAN Procedures	-	FORP
Relocatable Elements	-	REL
Absolute Elements	-	ABS

The '(CYCLES)' will show the number of symbolic cycles accumulated. This TOC will not include any deleted elements. An 'L' option is available which will produce the usual @PRT,T command format on demand stations when it is given with the 'T' option. If the 'L' option is not given on a @PRT,T, the short version of the TOC will be produced.

NOTE: A period must follow the filename or else the standard dropout rules apply and the specification will be considered to be an element of the file TPF$.

b) The TOC information for all elements of the name specified are printed in the usual TOC format. The format for this command is @PRT,T FILENAME.ELEMENTNAME, and applies to demand or on-site runs.

NOTE: The TOC information is taken from the element table item entry.

5.4.6 @PCH

The @PCH statement is used to punch program file elements on 80-column cards (026 code).

5.4.6.1. Formatting the @PCH Statement

The format of the @PCH statement is:

@PCH,OPTIONS SPEC1,SPEC2

The allowable options for the @PCH statement are:

A,R,S	An element of a program file is to be punched on 80 column cards. SPEC1 names a program file on Fastrand and the name of the element to be punched. The options designate which types of the element are to be punched. Any combination of A,R,S may be given, but at least one option must be given. The element, by name and types given, must exist in a non-deleted state.

The elements punched will contain the necessary control cards to reinsert them in a program file. The first card of procedure elements will be a @PDP control card. The filename referred to on the control card will be the same as the file from which the element is punched.

Several other options are available that apply only to symbolic elements. The S option must be included or they will be ignored. The options are:

G	The input card images are to be compressed.
H	The input images are to have columns 73 through 80 overlaid with a sequence number. SPEC2 should contain up to three alphanumeric characters. The characters will be left adjusted and overlay columns 73 through 75 of the input images.
J	The input images are to be compressed. The output images are to be sequenced in columns 73 through 80.

The G and J options may not both appear on the same statement. Sequence numbers are 100 (decimal) apart. Relocatable and absolute elements are sequenced automatically in columns 79, 80. The sequencing starts with AA and ends with ZZ. It is repeated if necessary. The compressed form of punched output uses a space count to strip spaces from the input images.

5.4.6.2. Examples of the @PCH Statement

```
@PCH,S TPF$.RUNPROG
```

The symbolic element RUNPROG in the program file TPF$ is punched on 80-column cards, one image per card.

```
@PCH,SRJH A.B,XYZ
```

The symbolic element B in program file A is punched on 80-column cards. The input images are sequenced in columns 76-80. The identification field XYZ is placed in columns 73-75. The sequenced images are compressed in columns 1-72 on the punched cards, and columns 73-80 are then sequenced. The relocatable element B in program file A is punched. The text has been previously sequenced by the assembler or compiler. FURPUR sequences the preamble. If the symbolic element was a procedure element, the deck should be used as input to @PDP, otherwise to @ELT. The relocatable deck may be used as input to @ELT.

5.4.7. @CHG

The @CHG command may be used to change the name of elements in a program file. The file must be catalogued or assigned to the run.

The format of the @CHG statement is:

```
@CHG,OPTIONS SPEC1,SPEC2
```

One or more of the A, R, and S options must be specified. SPEC1 names a program file and an element name. SPEC2 names the same program file and the new element name. The element name subfields include the element name and version. Element cycle may not be specified. Only those types specified by the options will have their names changed.

5.4.7.1. Examples of the @CHG Statement

```
@CHG,S      FILE1.ELT2,FILE1.ELT5/VERS3
```

Change symbolic element name ELT2 of program file 'FILE1' to element name 'ELT5' and add a version name 'VERS3.'

5.4.8. @PACK

The @PACK statement is used to rewrite an entire program file so as to exclude deleted elements and their associated entries in the table of contents. The entry point table is destroyed.

The format of the @PACK statement is:

```
@PACK SPEC1,SPEC2,..,SPECN
```

Each specification field given must name a program file on Fastrand to be @PACK'ed. All granules no longer needed to contain the file are released to the system.

5.4.9. @PREP

The @PREP statement is used to create an entry point table from the preambles of the non-deleted elements of a program file. If a previous entry point table existed, it is destroyed prior to creating the new one. The entry point table is required in some cases for use by the COLLECTOR. The entry point table may be listed by @PRT,T.

The format of the @PREP statement is:

```
@PREP  SPEC1,SPEC2,...,SPECN
```

Each specification field given should name a program file on Fastrand to be @PREP'ed.

5.4.10. @ERS

The @ERS statement is used to return all granules allocated to a program file back to the system.

The format of the @ERS statement is:

```
@ERS SPEC1,SPEC2,..,SPECN
```

Each specification field given should name a program file on Fastrand to be @ERS'ed.

5.4.11. @REWIND

The @REWIND statement is used to rewind tapes. The format of the @REWIND statement is:

```
@REWIND,OPTION SPEC1,SPEC2,...,SPECN
```

Each specification should give the name of a tape file. Each tape referred to by filename will be rewound. The only option allowed on this statement is the 'I' option. If the I option is present, the tape will be rewound with interlock.

A @REWIND,I does not free the tape drive for use by other runs. The statement

```
@FREE   FILENAME
```

performs a rewind with interlock and, in addition, makes the tape drive available to other runs. @FREE should be used rather than @REWIND,I whenever possible.

5.4.12. @MARK

The @MARK statement is used to write a hardware end of file mark on magnetic tape. The function is accomplished by writing two end of file marks and backspacing over the second one.

The format of the @MARK statement is:

```
@MARK SPEC1,SPEC2,...,SPECN
```

Each specification field given must contain a filename that refers to tape.

5.4.13. @CLOSE

The @CLOSE statement is used to write two hardware end of file marks on a tape and then rewind the tape.

The format of the @CLOSE statement is:

```
@CLOSE,OPTION SPEC1,SPEC2,..,SPECN
```

Each specification field given must contain a filename that refers to tape. The I option is the only option allowed. The I option indicates the tape is to be rewound with interlock.

Whenever possible, the statements

```
@MARK FILENAME
@FREE FILENAME
```

should be used rather than a @CLOSE,I. This will allow other runs to use the tape drive.

5.4.14. @MOVE

The @MOVE statement is used to move a magnetic tape over a specified number of file marks.

The format of the @MOVE statement is:

```
@MOVE,OPTION SPEC1,N
```

SPEC1 must contain a filename that refers to tape. N is the number of EOF marks to move past. B is the only option available with the @MOVE statement. The tape is moved forward without the B option, backward with the B option.

5.4.15. @FIND

The @FIND statement is used to locate an element on tape in element file format and position the tape immediately before the label block of the element. @FIND searches forward until the element is found, or until an end of file is encountered. In the latter case, the tape is backspaced to the previous end of file mark and the search is repeated. Encountering an end of file this second pass is an error exit for the function. Normally the @FIND statement is used just prior to a @COPIN statement requesting the element just located to be inserted in a program file on Fastrand or to insert all those read up to the next hardware end of file mark.

The format of the @FIND statement is:

```
@FIND,OPTION SPEC1
```

One and only one of the options A,R,S must be given to specify the type of element to be located. SPEC1 names a file on tape and the element to be located.

5.4.16. @CYCLE

The @CYCLE statement is used to set the maximum number of F-cycles to be retained for a given filename existing in the directory, or to set the maximum number of cycles to be maintained for a program file symbolic element on Fastrand. Procedure elements may not have their maximum changed (one cycle). The original maximums are 32 for F-cycles and 5 for element cycles.

5.4.16.1. Formatting the @CYCLE Statement

The format of the @CYCLE statement is:

```
@CYCLE SPEC1,N
```

The function performed is determined by the contents of SPEC1. There are two cases:

(1) SPEC1 names a catalogued file. In this case the statement applies to the number of F-cycles. The filename and the F-cycle specified must both be in the directory. N specifies the new maximum number of F-cycles to be retained. If N is 0 the entire F-cycle series and the filename entry will be dropped. If the current number of F-cycles for this filename is greater than the new maximum, the drop flag will be set in those F-cycles, starting with the oldest, necessary to satisfy the new maximum. Both security keys, if the file was catalogued with both, must be given when the file is originally assigned to the run. None of the F-cycles may currently be assigned exclusively to another run, or an error exit will be taken.

The number of F-cycles is taken to be the range of the newest and oldest absolute F-cycles.

(2) SPEC1 contains the name of a program file on Fastrand and the name of a symbolic element in the file, other than a procedure element, that is to have its maximum changed for number of cycles to be retained. N is the new maximum number to retain. If the number of cycles currently retained is actually greater than the new maximum, a new element with the same name will be created with those oldest cycles of the element eliminated as necessary to satisfy the new maximum. The old element is marked deleted.

5.4.16.2. Examples of the @CYCLE Statement

```
@CYCLE Q*A.B,2
```

Suppose the symbolic element B in program file Q*A originally contained 4 cycles, cycles 5,6,7,8. The new maximum requires that a new element B containing only cycles 7 and 8 be created. The old element B is deleted.

If the new limit were 5, only the parameter field of the element entry would be changed. No new element would be created.

```
@CYCLE Q*A.,2
```

Suppose the directory entry for the file Q*A indicates that 4 F-cycles exist with the name Q*A. Then the drop flag in F-cycles (-3) and (-2) will be set.

5.4.17. @ENABLE

The @ENABLE statement is used to remove the disable flag, which may have been set by the executive as the result of some type of malfunction, from a catalogued file in the directory. Under normal operation the user will not need the @ENABLE statement in his run.

The format of the @ENABLE statement is:

```
@ENABLE SPEC1,SPEC2,...,SPECN
```

Each specification field should contain the name of a catalogued file that is to be @ENABLE'd. A message will be printed if the file was not previously disabled. A normal exit is taken whether the file was disabled or not.

5.5. Multireel Files

Some provisions exist in FURPUR for the creation of tape files that extend over more than one reel. The @COPOUT function makes its own call to TSWAP$ in the event an end of tape condition is encountered. @COPOUT writes a 14 word end of reel sentinel which is understood by @COPIN as an indication that the element being read extends onto a second reel. The REWIND statement returns a user to the first reel of tape assigned the given filename. @COPY,F and @COPY,I also permit the reading and writing of multireel files. End of reel sentinels created by functions other than FURPUR may not be interpreted correctly by FURPUR. The @MOVE statement checks for end of reel sentinels created by FURPUR.

6. SYSTEM PROCESSORS

6.1. The COLLECTOR (@MAP Processor)

6.1.1. General

The COLLECTOR for the UNIVAC 1108 is designed to provide the user a straight-forward means of collecting and interconnecting relocatable elements to produce a program which is in a form ready for execution under control of the 1108 executive system. This program form is called an absolute element. Internal references are linked together, and no modification is necessary to load the program anywhere in core. Optionally, the COLLECTOR can be used to produce one relocatable element from a collection of several relocatable elements.

The COLLECTOR is concerned with three basic inputs. These are:

(1) Parameters from the executive control statement causing the collection.

(2) Source language control statements.

(3) Relocatable elements from a variable number of sources.

All of these inputs are discussed in detail later; however, a brief description of each is given here for introductory purposes.

Basically, the COLLECTOR is called whenever a @MAP executive control statement is encountered within a control input file. The @MAP statement specifies the input and output elements to the COLLECTOR as it does for other system processors. The information contained within the @MAP control statement is comprehensive enough to direct the allocation of most programs.

For performing 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 source language control statements. These statements may follow the @MAP executive control statement or be contained in an element in a program file.

Complete capabilities are available through the COLLECTOR for updating the symbolic element in the program file. The procedure is the same as for updating any other source language (FORTRAN, COBOL, etc.) element processed by the system.

Relocatable elements to enter into the collection are indicated to the COLLECTOR by way of the two input sources just described. Relocatable elements from libraries may be specifically named to be included in the program or included only if an external reference is made to the element. Generally, all the relocatable elements in the temporary program file (TPF$) are arbitrarily included in the program being collected. Use of the system relocatable library to satisfy external references is automatically implied; the use of user libraries is under control of a source language control statement to the COLLECTOR. Any specified user libraries are always searched before the system relocatable library.

The outputs of the COLLECTOR are as follows:

(1) An absolute or relocatable element.

(2) A source language control element as discussed above.

(3) Listing information

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.

Normally, the COLLECTOR includes within an absolute element, a set of tables for use by the diagnostic system. As discussed later, this output can be suppressed by the user.

For any error condition encountered, the COLLECTOR produces an error message which is placed in the print file assigned to the run (PRINT$).

The ensuing sections describe in detail the executive control statements involving the COLLECTOR, the source language control statements processed by the COLLECTOR, the operational characteristics of it, and procedures for segmenting a program.

6.1.2. Executive Control Statements

6.1.2.1. The @MAP Control Statement

The @MAP executive control statement is used for specifying that the COLLECTOR is to be used to combine a set of relocatable elements into a

single absolute element or into a single relocatable element. The @MAP statement has the same format as other processor statements and is:

```
@MAP,OPTIONS  SI, RO, SO
```

All standard processor options (see the section entitled Statement Format for Language Processors) are legal on the @MAP statement. These include A, I, L, N, P, Q, S, U, W, and X. Specific action taken on print options is as follows:

L - Produce complete listing containing all information about the core space used by the program, the space allocated to each element, the program address of all definitions, the external references of each element, and the scale drawing of segmentation.

S - Produce a summary listing, and include the scale drawing of program segmentation.

N - Produce no listing, except diagnostics.

The MAP ID line (line a below) will always occur regardless of originating site or options on the @MAP card. The START line (line b below) will appear on demand runs when there are no options on the @MAP statement or when @XQT is used to call and initialize the COLLECTOR. For batch runs or when option(s) are present on the @MAP control image, but do not include any listing options (L, S, or N), the COLLECTOR will assume an 'S' option.

a. MAP XXXX-MM/DD-HH:MM (IC,OC) where:

   XXXX level number of current COLLECTOR

   MM/DD - current month and day

   HH:MM - time of day in hours and minutes

   IC - input cycle number, if any

   OC - output cycle number, if any

b. START = SSSSSS, PROG SIZE (I/D) = II/DD

   where:

   SSSSSS - program's starting address in octal

   II - total number of I-bank words in decimal

   DD - total number of D-bank words in decimal

The following options, unique to the @MAP processor, may also be specified.

Z	Inhibit generation of the diagnostic information normally provided to the diagnostic system.
R	Produce a relocatable element rather than absolute element. (Entry points in the relocatable elements being combined in an R-option collection, which are still to be defined as entry points in the single relocatable output element, must be named on the COLLECTOR DEF statement. The DEF statement is explained below).
D	Give a diagnostic message for all possible address fields over 65K. Error checking of possible instruction format violations is done.
F	Set quarter word PSR bit.
T	Do not set quarter word PSR bit. If neither F nor T is given, the setting will be determined by the sensitivity of the element containing the starting address.
E	Allow program to exceed address 0177777 (decimal 65K). If the E option is not present and a programs D-bank exceeds 65K, the D-bank starting address is pushed down toward the last I-bank in order to fit the D-bank code under 65K. The presence of the E option indicates that the user knows that his program may exceed 65K and has made special programming considerations to handle this. Minimum address specifications are not violated in any case.
B	Set bit in absolute element's header table indicating to the loader that the program area need not be zero-filled prior to loading. In addition, core acquired by an ER MCORE$ will not be cleared.

'SI' is 'FILE1.ELT1/VERS1(CYCLE)'. It normally identifies the symbolic input element. When the I option is used, it identifies the symbolic output element.

'RO' is 'FILE2.ELT2/VERS2'. It identifies the absolute output element of the collection. (If the R option is on, this is instead a relocatable element. The field is denoted 'RO' regardless of whether output is relocatable or absolute.)

'SO' is 'FILE3.ELT3/VERS3'. It identifies the (optional) symbolic output element when the I option is not used. If corrections follow, they

will be included in the new element. The cycle number of the new element is set to zero.

Standard system dropout rules apply to these 3 specification subfields. A single name (with no period) in any of these subfields is assumed to be an element in the run's temporary program file (TPF$). A single name with a leading period in 'RO' or 'SO' is assumed to be an element in the file given in the specification preceding. A single name with a leading period in 'SI' is assumed to be in TPF$.

Here are some examples of the @MAP statement:

```
@MAP  SYMIN/C,BACKUP.ABSOUT
```

(Element SYMIN version C, latest cycle, from TPF$, is used to direct the collection of element ABSOUT written into file BACKUP. If any corrections follow, they will be used but not saved, because no output symbolic element is produced. Only the ID and START lines are printed, for demand runs.)

```
@MAP,I  BACKUP.SYMOUT,.ABSOUT
```

(The statements following the @MAP statement are used to direct the collection and are output as element SYMOUT in file BACKUP. The absolute output element is also put into file BACKUP. A listing is produced as if an S option were specified.)

```
@MAP,U  SYMIN(3),ABSOUT/REVISED
```

(Cycle 4 of element SYMIN in TPF$ is produced, saving any corrections that follow the @MAP statement. Absolute output element ABSOUT, version REVISED, is also put into TPF$. A listing is produced as if an S option were specified.)

```
@MAP,I
```

(This is a special case where the system picks out its own name for absolute output element: the input symbolics (if any) are not saved. Both internal table entries, and the printed output produced by a @PRT,T statement following the collection will look as if the @MAP statement had contained: @MAP,I ,TPF$. NAME$ A listing is produced as if an S option were specified.)

```
@MAP
```

(This is treated as if it has an I option, which is the same as the special case above. However, only the ID and START lines are printed, for demand runs.)

```
@MAP,IRXLED  A,A
```

(The statements following the @MAP statement are used to

direct the collection and are output to TPF$ as symbolic element A. The result of the collection is output to TPF$ as relocatable element A. If errors of any kind are encountered, inhibit continuation of program. Produce a full listing. Allow DBANK to exceed 65K. Print diagnostics for address fields over 65K and possible bad instructions.)

```
@MAP  OLDFILENAME.OLDELEMENT,A,NEWFILENAME.NEWELEMENT
```

(The symbolic list of collector commands contained in OLDELEMENT as amended by any corrections following this @MAP statement, is used to direct the collection, and is output into a new file with a new element name. The absolute output element, A, goes to TPF$.)

These 3 examples follow one another in sequence, and show some actual COLLECTOR control statements (which are explained later on):

```
@MAP,I    F8.MAPSYM,.MYPROGRAM
LIB       F8.
SEG       CONTROL
IN        ELEMENT1
SEG       OVERLAY5,(CONTROL)
IN        F9.

@MAP,U    F8.MAPSYM,.MYPROGRAM
LIB       F10.
-3,3
IN        ELEMENT0
SEG       OVERLAY4,(CONTROL)
IN        F7.
-5
SEG       HIGHCORE,(OVERLAY4,OVERLAY5)
IN        DATAELEMENT
CLASS     REVISED
NOT       ELEMENT1
```

(after this second @MAP, element MAPSYM will save the corrections, because an update of cycle was specified. The absolute element MYPROGRAM put into file F8 by the first @MAP will be deleted and replaced by the different MYPROGRAM produced by this second @MAP.

```
@MAP,I    F8.MAPSYM,.MYPROGRAM
LIB       F10.
LIB       F8.
SEG       CONTROL
IN        ELEMENT0
SEG       OVERLAY4,(CONTROL)
IN        F7.
SEG       OVERLAY5,(CONTROL)
```

```
IN        F9.
SEG       HIGHCORE,(OVERLAY4,OVERLAY5)
IN        DATAELEMENT
CLASS     REVISED
NOT       ELEMENT1
```

(Element MYPROGRAM comes out exactly the same here as the preceding MYPROGRAM which it replaces in file F8. Element MAPSYM, set here to initial cycle 0, also comes out the same as the update cycle 1 of MAPSYM which it replaces in file F8.)

If a program file named on a @MAP statement (or on a 'LIB' or 'IN' COLLECTOR control statement) has not been assigned, but has been previously catalogued, it will be assigned automatically during the collection. At the end of the collection, it will be returned to its original state with a @FREE (plus X, R, and/or A option).

6.1.2.2. The @XQT Control Statement

For execution of an absolute program created by the COLLECTOR, the following control statement is used:

```
@XQT,OPTIONS  ELTNAME
```

Any options specified are available to the user's program by the OPT$ executive request whenever it is initiated. The field 'ELTNAME' specifies the absolute program to execute, in standard file/element notation. This field is the counterpart of 'RO' in the @MAP control statement. If no element is specified, the last absolute element placed in the file given (TPF$ if no file is specified) will be loaded and executed. In the absence of an absolute element, all of the relocatable elements in the file are both collected and executed (a relocatable element may not be explicitly named on an @XQT statement, however.)

Examples of the @XQT statement in typical run streams:

(1)
```
----      Compilations to produce relocatable elements in the
          run's temporary program file, TPF$.
@MAP,S    Generate an absolute element consisting of all the
          relocatable elements in the temporary file.
@XQT      Execute the absolute element generated above.
```

(2)
```
----      Compilations producing relocatable elements in the
          user specified file, FILEA.
```

```
@MAP FILEA.SYMBOLIC,FILEA.XYZ
```

Generate the absolute element XYZ in FILEA as directed by the source element FILEA.SYMBOLIC.

```
@XQT FILEA.XYZ
```

Execute the absolute element generated above.

6.1.3. COLLECTOR Control Statements

6.1.3.1. General

In addition to the information specified in the @MAP control statement, a set of source language control statements can be processed by the COLLECTOR to provide the user the capability of controlling the construction of even the most complex programs. The user can enter these control statements via his control input stream for each collection, or he can create within a program file a source language control element containing the statements. This element can be updated by entering the corrections via the control input stream.

The control statements recognized by the COLLECTOR include the following:

IN	Include specific elements in the collection.
NOT	Exclude specific elements from the collection.
LIB	Specify libraries to be searched.
REF	Specify the external references to be retained in the absolute or relocatable element.
SEG	Direct the segmentation of a program.
DSEG	Specify a dynamic segment.
RSEG	Specify a relocatable segment.
DEF	Specify external definitions to be retained in the absolute or relocatable element.
ENT	Specify the starting address of a program.
EQU	Give values to undefined symbols at the time of the collection.
CLASS	Specify a mask to use in selecting elements for collection.
COR	Specify that corrections are to be made to an element.
SNAP	Direct positioning of snapshot dumps.
END	End of source language statements to be processed.

6.1.3.2. The IN Statement

The IN control statement allows the user to include any or all elements from any number of files in his collection and specifically in the segment named by the preceding SEG statement. The format of this statement is the following:

```
IN   FILE1.ELT1/VER1,FILE2.ELT2/VER2,...
```

The fields 'FILE1.ELT1/VER1,' etc., identify specific elements to be included in the collection. By specifying only 'FILE1.' the user can specify the inclusion of all elements in a program file.

Normally all the relocatable elements in the run's temporary program file TPF$, are included in every collection. If the external definitions of the temporary file have been collected with a @PREP control statement, the elements are included selectively. In this case TPF$. is the first program file examined for element inclusion. Elements that are not associated with files may be included from TPF$, or any program file named in LIB statements.

An element name may appear on only one IN statement and only once in a collection. It is important, for FORTRAN programmers especially, to note that no COMMON block name (labelled, or named common) in the collection may be identical to any element name. Throughout most of the collection common blocks and elements are handled in a very similar manner, and their names must distinguish them from one another.

The following are examples on the use of the IN statement:

```
IN    FILEA.,FILEB.
      (All relocatable elements in FILEA and FILEB are included.)

IN    FILEB.BB,.CC,DD
      (elements BB and CC from FILEB, and element DD, whose file is
      not indicated, are included in the collection.)
```

A whole file may be named on an IN statement and individual elements in the file may be named on additional IN statements, possibly in different segments of the collection. The elements explicitly named are included as specified, and the whole file IN serves to subsequently bring in only the balance of the file. For example, suppose file MAKE contains elements named 0001 through 1500. The following commands position element MAKE.0733

in segment MAIN, and the remaining elements, 0001 through 0732 and 0734 through 1500, in segment OVERLAY:

```
SEG     MAIN
IN      MAKE.0733
SEG     OVERLAY
IN      MAKE.
```

6.1.3.3. The NOT Statement

The NOT control statement is essentially the inverse of the 'IN' statement. It allows the user to state explicity which elements within files are not to be included in a collection. The format of this statement is as follows:

```
NOT   FILE1.ELT1/VER1,FILE2.ELT2/VER2,...
```

Where the successive fields indicate elements not to be included. If the version name or file name is omitted, all elements with the specified name are bypassed.

The following are examples on the use of the NOT statement:

(1)
```
@MAP,I  A,A
NOT  AA,BB
          (All relocatable elements in the TPF$ except AA and BB
          are included in the collection.)
```

(2)
```
@MAP,I  A,A
IN  FILEA.
NOT  FILEA.AA,.BB
          (All relocatable elements in FILEA except AA and BB are
           included.)
```

(3)
```
@MAP,I  A,A,
IN  FILEA.,FILEB.
NOT  FILEA.AA,.BB,FILEB.CC,.DD
           (All relocatable elements from FILEA except AA and BB, and
            all relocatable elements from FILEB except CC and DD are
            included.)
```

The NOT statement is honored for whole files, and overrides any other explicit or implicit specifications to wholly or partially include, or search any file or files. As with the IN statement, a period must follow the file name to show that this is not an element being named.

```
EXAMPLE:  NOT   TPF$.,  SYS$*RLIB$.
```

6.1.3.4. The LIB Statement

The LIB control statement allows the user to specify libraries to be serached by the COLLECTOR for the purpose of satisfying external references and/or finding elements specified without file names. The format of the LIB statement is:

```
LIB  FILE1,FILE2,...
```

The names of files to be treated as libraries are specified in successive fields. These libraries are searched in the order in which they are given and before the system library (SYS$*RLIB$) file. Files may be specified to be searched more than once by naming them more than once in a LIB statement. Several LIB statements may be specified and their effect is cumulative. Files are not searched for external definitions if the file has not been prepared by the FURPUR @PREP statement.

Typical LIB control statements are as follows:

```
LIB USER1    (File USER1 is searched before the system library).

LIB USER1,USER2
             (File USER1 and then file USER2 are searched before
              system library).
```

6.1.3.5. The SEG Statement

The SEG control statement is used to inform the COLLECTOR of the beginning of a new segment in those programs requiring segmentation. The format is:

```
SEG  NAME1,NAME2
```

or

```
SEG  NAME1,.(NAME2,NAME3,...)
```

The field 'NAME1' is the name of the segment and must be specified. The field 'NAME2', etc. gives the names of other segments to which the segment 'NAME1' is being related. A segment can be specified for automatic (indirect) loading when referenced by suffixing an asterisk to 'NAME1': 'NAME*'.

If the field 'NAME2' is void, the segment being specified is originated immediately following the segment defined by the preceding SEG statement. The

field 'NAME2' (not included in parentheses) specifies that the segment being defined is to originate at the same location as does segment 'NAME2'. If the right hand field contains one or more segment names enclosed in parentheses, the segment 'NAME1' is started following the highest address occupied by any of these segments.

Each segment may have two program areas (banks), namely, the instruction area and the data area (also referred to as IBANK and DBANK). Therefore, a segment specified to follow the highest address of several segments may have its instruction area follow the instruction area of one segment and its data area follow the data area of a different segment.

The first segment named in the source input is called the main segment and is not overlayed by other segments.

Segments may be loaded and executed independently of one another. However, the placement of elements common to several segments may dictate that some segments must be in memory when others are being executed. Elements are not necessarily attached to the main segment when they are referenced in more than one segment but not explicitly included in any segment. Each segment has a path leading to the main segment. Elements referenced by two (or more) segments are attached to the segment that is in the path of all the referencing segments. Named common blocks are likewise in the path of all segments referencing the block.

The path to the main segment follows the path of the first segment in its path. The first segment in its path is determined by its relation specifications.

```
SEG  A,(B)
  or
SEG B
SEG A
```

Segment A's path starts with segment B and follows B's path to the main segment.

```
SEG A,B
```

Segment A's path is identical to segment B's

```
SEG A,(B,C,D)
```

The first segment in A's path is the segment common to the paths of segments B,C and D.

At least one IN statement must follow the SEG statement.

All the elements specified on IN statements after this SEG statement and before the following SEG statement are a part of this segment. Other elements referenced are included in this segment or in a segment in its path to the main segment.

6.1.3.6. The DSEG Statement

The DSEG control statement is used to inform the COLLECTOR of the beginning of a segment with special characteristics. This type of segment is called a dynamic segment. The core area of the segment in excess of normal segments may be temporarily released to the executive system with a reference to DREL$. The area is released automatically when a dynamic segment is overlayed. The area is released only when it is at the end of the program's area. Since the executive system may need to move other programs to load a dynamic segment, discretion should be used in designating what segments are dynamic segments.

The DSEG statement has the same format as the SEG statement.

6.1.3.7. The RSEG Statement

The RSEG statement is used to inform the COLLECTOR of the beginning of a segment that is relocatable. The segment contains only an instruction area. Relocation of address fields is accomplished by adding the beginning address of the segment to the right or left half of the words to be relocated. Relocatable segments may not be designated for indirect loading. The elements to make up a RSEG segment must be explicitly named on IN statements following the RSEG statement.

6.1.3.8. The DEF Statement

The DEF control statement is used to list those external definitions to be retained by the resulting absolute or relocatable element. For absolute elements, the program may be entered by interpretive code (output of the conversational processors) at any of the external definitions listed. The address of this table is defined by the COLLECTOR to be ENTRY$. It is addressable by the program using the tag ENTRY$. Also, DEF and/or REF statements cause the COLLECTOR to build a table defining the labelled COMMON blocks in the program. This table is addressable by the COLLECTOR defined tag COMMN$. The format

of the statement is:

```
DEF   DEF1,DEF2,DEF3,...
```

Where the successive 'DEFi' fields are the names of external definitions to be retained. An example of this statement is as follows:

```
DEF   SIN,COS,SQRT
```

(The listed external definitions are retained by the resulting element.)

6.1.3.9. The REF Statement

The REF control statement is used to list those external references to be retained by the resulting absolute or relocatable element. For absolute elements, the external references listed may be linked to interpretive code by the interpreter. The table is addressable by the COLLECTOR defined tag XREF$. The format of this statement is as follows:

```
REF   REF1,REF2,REF3,...
```

Where the successive 'REFi' fields are the names of the external references to be retained. No attempt is made to satisfy these references from either user libraries or the system library. An example of the REF statement is:

```
REF   SIN,COS,SQRT
```

(The listed external references are retained by the new element)

6.1.3.10. The ENT Statement

The ENT control statement provides the user the capability of overriding the starting address specified via the entrance to a main program generated by FORTRAN, COBOL, ALGOL, etc. The format of this statement is:

```
ENT NAME
```

Where NAME is an externally defined symbol. Control is transferred to the absolute location generated for this symbol whenever the program is subsequently executed. In the absence of an ENT statement, the starting address will be taken to be a transfer address encountered in the processing of relocatable elements. The starting address must be in the main segment.

6.1.3.11. The EQU Statement

The EQU control statement may be used to give a value to an undefined symbol at the time of collection. The format of this statement is:

```
EQU   NAME1/VALUE1,NAME2/VALUE2,...
```

Where 'NAME1' is a symbol to be defined and 'VALUE1' is the value to be given to the symbol. The same is true for 'NAME2/VALUE2', etc. Each subfield 'VALUEi' may be an octal or decimal integer, a symbol, or a symbol with an offset. If a symbol is used, it must be externally defined by one of the elements to be included. Examples of the use of the EQU statement are as follows:

(1) EQU JOE/0200
(External reference JOE is defined to be 0200.)

(2) EQU AL/SAM+10
(External reference AL is defined to be SAM+10; SAM must be externally defined.)

(3) EQU JOE/0200, AL/SAM+10
(Same as 1 and 2)

6.1.3.12. The CLASS Statement

The CLASS statement may be used to specify the relocatable element to be included in the collection when otherwise more than one element could qualify. There are two conditions where more than one element may qualify:

(1) The version is not specified on an IN statement and more than one relocatable element has that same name.

(2) More than one relocatable element defines an external reference and none of the elements has been explicitly included in the collection, or all but one explicity excluded from the collection.

The format of the class statement is:

```
CLASS   STRING
```

Where the field 'STRING' is twelve characters including the special character asterisk (*). The asterisk designates a character position that is to be ignored when making comparisons. When several elements qualify to be included in the collection, the COLLECTOR compares this string of characters with the

versions of the qualifying elements. If the element does not have the same characters in the version as the characters of the 'STRING' (for each character position), it no longer qualifies for inclusion.

When only one element remains qualified after the comparisons, that element is included in the collection. When more than one element still qualifies, the versions of these elements are compared to the character string of the next CLASS statement. If more than one element qualifies after the CLASS statement parameters have been exhausted a diagnostic message is given. None of the 'qualifying' elements is included in the program since a unique element may be found in the next library examined. It should be noted that different orders of CLASS statements may give different results.

Assume that the element named SIZE is named on an IN statement and the following relocatable elements are in the temporary library:

```
SIZE/A2SMALL
SIZE/B3LARGE
SIZE/D3SMALL
SIZE/D2LARGE
```

The source language to the COLLECTOR is:

```
@MAP,I S,A
SEG  AA
IN  SIZE
CLASS  **LA********
CLASS  D**********
```

The element SIZE/D2LARGE is included in the collection. The one class statement:

```
CLASS   D*LA*********
```

will give the same results.

6.1.3.13. The COR Statement

The COR statement is used to specify that relocatable corrections are to be made to an element included in the collection. The format of the COR statement is:

```
COR   ELT
```

The relocatable corrections for the element 'ELT' follow the COR statement.

Relocatable corrections may be one of three formats:

```
ADDRESS,LC1 F J A X H I U,LC2,ELT1
ADDRESS,LC1 DATAWORD
ADDRESS,LC1 DATA,LC2 DATA,LC2
```

The field 'ADDRESS' specifies the relative address under location counter 'LC1' to make the correction. The F, J, A, X, H and I fields correspond to portions of the UNIVAC 1108 instruction word. The fields 'U' and 'DATA' may be a symbol, symbol and offset or an octal or decimal number. Octal numbers require a leading zero. The field 'DATAWORD' must be numeric. The optional field 'LC2' indicates that the 'U' or 'DATA' fields are relative to the value of the location counter 'LC2.' The optional field 'ELT1' specifies the element in which 'LC2' belongs, if it is other than the element being corrected. The 'DATA' fields represent the upper and lower halves of the word.

COR statements are bypassed in an R-option collection.

6.1.3.14. The SNAP Statement

The SNAP control statement specifies elements in which snapshot dumps are to be taken. The format of the statement is:

```
SNAP  ELT
```

The field 'ELT' is an element included in the collection. Statements following the SNAP statement give the parameters for the snapshot dump The format is:

```
ADDRESS,LC1  ADDRESS  LENGTH,R  TIMES,FREQUENCY
```

The field 'ADDRESS,LC1' specifies the address of the instruction to be replaced with a dump request. The field 'ADDRESS' gives the address to start the dump. The field may be 'U,LC2,ELT1' or symbol and offset as in the COR statement parameters. The field 'LENGTH' specifies the length of the memory area to dump. The field 'R' is used to indicate which of the registers is to be printed according to the values:

R =

0-no registers
1-R registers
2-A registers
3-A and R registers
4-X registers
5-X and R registers
6-X and A registers
7-X, A and R registers

The 'TIMES' field specifies the number of times the snapshot is to be taken. If omitted, the value is 100. The field 'FREQUENCY' specifies at what intervals the dump is to be taken. The value three specifies the dump to be taken every third reference; five every fifth reference, etc. The value of one is assumed if the field is omitted.

At most sixteen snapshot parameter statements may be included in one collection.

What actually happens with a snapshot dump request is that an 'SLJ SNAP$$' instruction is inserted at the location at which the SNAP is called. SNAP$$ is an entry point in element SNAP$. The replaced instruction word is saved in a table in element SNAP$. 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, control goes immediately to the jump designation. Otherwise, control is transferred to the location following the location at which the SNAP is called.

Because of this execution of the replaced instruction from within SNAP$, the replaced instruction must not be -

- Altered during the course of program execution.
- Referenced as data.
- Referenced by indirect addressing.
- an SLJ which specifies indirect addressing or indexing.
- an LMJ which specifies indexing.
- an EX which indirectly references an LMJ or SLJ.
- a TEST skip instruction.

6.1.3.15. The END Statement

If a data statement (as opposed to a control statement -- @ in column 1) which is unrelated to the collection follows a @MAP statement, an END statement placed after the last COLLECTOR source statement may be necessary to tell the COLLECTOR to disregard the data statements following. The format of the statement is:

```
END
```

6.1.4. Functional Aspects

Functionally, the COLLECTOR must interpret the source input language, find the elements to include, and generate the output listing. The following description pertains to the procedure for the more general case of a segmented program; however, a non-segmented program can be considered as being a segmented program with only a main segment.

Initially, parameters from the @MAP (or @XQT) control statement are obtained and interpreted. All of the COLLECTOR control statements are interpreted and saved in tables internal to the COLLECTOR. Diagnostic messages are given where appropriate. Elements named on IN statements but not preceded by a file name are maintained in a list apart from elements named with a designated file name.

Elements are added to the collection in two steps. The first step involves finding the elements explicitly named on IN statements, and processing the information contained in the preamble section of each element. In addition to the explicitly named elements, all the elements in the run's temporary program file (TPF$) may be automatically processed in step one. When TPF$ has been prepared by the @PREP statement (a blank name field on a @PREP statement always implies TPF$), the automatic processing of its elements is inhibited. TPF$ elements automatically processed are added to the program only if references are made to them, or if there are no IN statements at all (which the COLLECTOR treats as an implied: IN TPF$).

TPF$ is always the first library searched for elements explicitly named but without file designation. (the COLLECTOR includes TPF$ as the first entry in its internal LIB table, as if there had been a LIB TPF$ statement). Files actually named on LIB statements make up the second level of searching. The last level of searching covers SYS$*RLIB$ (See EXECUTIVE CONTROL LANGUAGE CHAPTER: SYSTEM LIBRARIES). SYS$*RLIB$ is @PREP'ed at system boot time, and the COLLECTOR includes it automatically as the last entry of the interal LIB table, except in an R-option @MAP.

The preamble of an element includes the definition of each entry point in the element, the length of each location counter used, every symbol yet undefined in the element, and common blocks defined by the element. In processing the preambles, the entry points of the element are put into the internal

collector EP table. The undefined symbols are linked to an entry point by the same name in the EP table, or added to the list of symbols yet undefined (the UNDE list) from elements previously processed. Symbols are removed from the UNDE list as entry points of the same names are encountered in processing preambles.

The second step in adding elements to the collection involves searching libraries for elements with entry points of the same names as those in the UNDE list, accumulated in processing preambles of included elements. Only libraries that have been @PREP'ed (entry point table prepared) will be searched in the second step. Every undefined symbol currently in the UNDE list is looked for in each library. When an element is found with an entry point by the same name, the preamble of that element is then also processed as described above. The UNDE list is processed from top to bottom with new symbols added to the bottom. TPF$ is searched first, then files named on LIB statements, and then SYS$*RLIB$.

The order of the appearance of user-specified elements in any segment of a program will be the same as that in which they were specified so long as each was specifically named on an IN statement. When all elements within a file are included in a segment, the ordering of the elements within a group so specified will be random. An element included by a library search appears immediately preceding the user specified elements of the segment in the path of all segments referencing the element.

The most efficient collection results when every element desired in the collection is explicitly named, including file name. The reason for this is the @PREP requirements and library searches are eliminated.

The first address of the instruction area is assigned 01000 (octal). The address of the data area is always greater than the highest address of any reentrant processor in the system. Odd location counters of an element (1,3,5 etc.) are assigned to the instruction area. Even numbered counters (0,2,4 etc.) are assigned to the data area. Blank COMMON is assigned to the data area of the main segment. A named COMMON block is attached to the segment (if not named on an IN statement) in the path to the main segment of all segments referencing it.

Symbolic names of external definitions and external references (see

comments on the COLLECTOR DEF and REF statements), segment names, qualifier names, file names, element names, version names, and common block names, insofar as COLLECTOR restrictions are concerned, may be up to twelve characters in length and may contain any combination of alphabetic, numeric, $ or hyphen characters.

6.1.5. COLLECTOR Defined Symbols

In addition to the COLLECTOR defined symbols ENTRY$, COMMN$ and XREF$, Three more symbols are avilable to the user. The tag LASTD$ is given the value of the last address of the data area at collection time. Likewise, LASTI$ is the last address of the instruction area. The address of the segment load table is made available to the diagnostic system by defining the tag SLT$.

6.1.6. Program Segmentation and Loading

The following example is given to illustrate the use of the segmentation facilities of the COLLECTOR.

Assume FILEA has the following relocatable elements with the indicated references outside of the file:

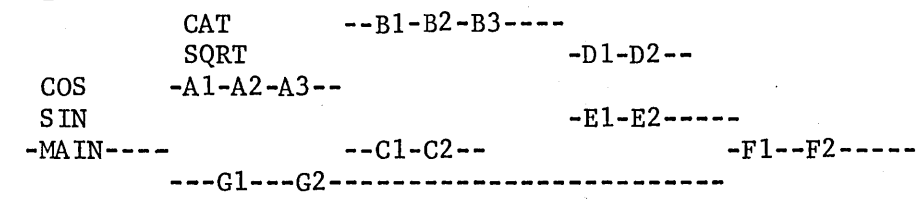
FILEA elements NAME/VERSION	References outside of FILEA required FILE.NAME/VERSION
MAIN	FILEA.A1,B1,F1
A1/A	
A2/A	LIB1.SIN/X
A3/A	LIB2.COS/X
B1/B	LIB1.SQRT/X
B2/B	
B3/B	
C1/C	LIB1.SQRT/X
C2/C	
D1/D	LIB2.CAT/Y
D2/D	
E1/E	LIB2.CAT/Y
E2/E	
F1	
F2	
G1/G	LIB1.SIN/X
G2/G	LIB2.COS/X
G3/G	

A particular collection setup for segmenting a program from this file might be as follows:

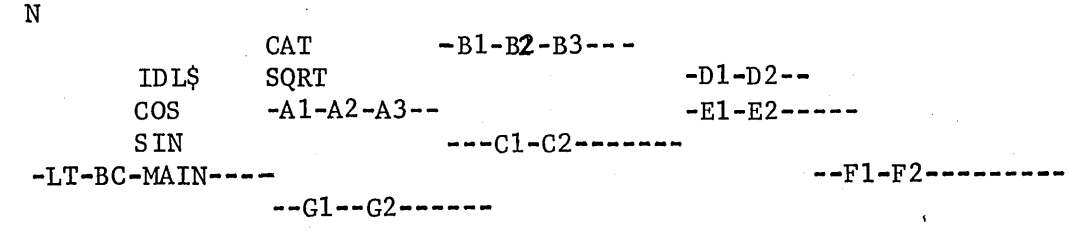
```
@PREP                          LIB1.
@PREP                          LIB2.
@MAP,LI                        MAPSYM,MAPABS
SEG                            MAIN
IN                             FILEA.MAIN
SEG                            A*, (MAIN)
IN                             FILEA.A1/A,.A2/A,.A3/A
SEG                            B*, (A)
IN                             FILEA.B1/B,.B2/B,.B3/B
SEG                            C*, B
IN                             FILEA.C1/C,.C2/C
SEG                            D*, (B,C)
IN                             FILEA.D1/D,.D2/D
SEG                            E*, D
IN                             FILEA.E1/E,.E2/E
SEG                            F*, (D,G)
IN                             FILEA.F1,.F2
SEG                            G*, (MAIN)
IN                             FILEA.G1/G,.G2/G
LIB                            LIB1,LIB2
@XQT
```

This particular set of control statements would result in the memory structure illustrated below. The horizontal coordinate is used to denote increasing memory addresses from left to right. Segments with common horizontal coordinates may not be in memory simultaneously.

INSTRUCTION AREA MEMORY MAP



DATA AREA MEMORY MAP



IDL$ is the name of the indirect load routine and is always in the main segment.

LT represents the segment load table and indirect load table generated

by the COLLECTOR.

BC stands for blank COMMON.

Note that the element CAT is attached to segment A and not segment B or segment C.

The first address of the data area, N, is greater than or equal to the mimimum data area address specified at system generation time. This enables the data area to be linked to reentrant processors. N is always a multiple of 01000.

6.2. The Procedure Definition (@PDP) Processor

6.2.1. General

The PROCEDURE DEFINITION PROCESSOR (PDP) accepts source language defining 1108 FORTRAN or COBOL procedures and builds an element in the user defined program-file. These procedures may subsequently be referenced in a compilation without definition.

One tablewill be generated for each type of procedure (FORTRAN,COBOL) in a program file. This table will contain any labels that are defined externally to the procedure. In the case of FORTRAN and COBOL procedures, these will be the labels on the proc line. 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 (FORTRAN, COBOL) has the same label an entry will be made in the table for each procedure, but a call on that procedure will produce the last one entered.

The PROCEDURE DEFINITION PROCESSOR is called whenever a PDP executive control statement is encountered. The format of this statement is as follows:

```
@PDP,OPTIONS SI,SO
```

The field 'OPTIONS' may contain any of the standard processor option letters plus any of the following letters to indicate directions to the procedure definition processor:

F	Indicate a FORTRAN procedure element.
C	Indicate a COBOL procedure element.

When the 'F' option is present, PDP assumes it is inserting or updating a FORTRAN procedure element. When the 'C' option is present, the PDP assumes it is inserting or updating a COBOL procedure element. The L option causes

a listing of the output element. The N and S options are ignored.

The fields 'SI' and 'SO' are of the standard format for symbolic element description. The field 'SI' normally identifies an input element by file, element name, version, and cycle. However, when the I option is used, 'SI' is the identification to be given to the new program file element.

The field 'SO' is used as the identification of a new output element whenever it appears. Standard system dropout rules apply to both 'SI' and 'SO'.

The I option is used solely to introduce source card images into a program file. When applying corrections to an element, the I option is not permitted.

PDP will permit processing procedural elements from a tape file that is in element file format. Furthermore, corrections to this element are permitted if a source element is produced in a program file. PDP will not attempt to interpret the names on a control card; i.e., it will make no effort to insure uniqueness or avoid possible duplication of names in 'SI' or 'SO' fields.

Cycle of procedures is permitted. The cycle number may be increased if the U option is on, but only one is retained, and when the procedure is called for inclusion, the latest cycle is given.

Some examples on the use of the @PDP statement are as follows:

@PDP,LC A.B,C	Produce a COBOL procedure element from file A (if it is PF or tape file) element B and put the new source code in File TPF$, element C.
@PDP,LC A.B,.C	Produce a COBOL procedure element from program file A element B, call it element C and put it in program file A. Produce a complete listing. Both elements B and C are retained. A tape file is not permitted for 'SO'. PDP will error out but will not abort unless the 'X' option is on.
@PDP,UF A.B	If file A is a program file element B has its cycle increased by 1, and the element is entered into the program file named 'A'. A may not designate a tape file.
@PDP,IF AFILE.PROC/AB	(Procedure definitions following the @PDP are introduced into AFILE as element PROC, version AB, cycle 0.)

@PDP,UC BFILE.PAT/DE	(Corrections are applied to element PAT, version DE, latest cycle, in BFILE to produce an updated cycle of the same element in the same file.)
@PDP,F AF.PR1,BF.PR2	(Any corrections following the @PDP statement are merged with the most recent cycle of element PR1 from file AF to produce cycle 0 of element PR2 in file BF.)

If PDP is processing elements from a tape file, the file must be positioned so the label block will be read in. If the name in the label block does not agree with the 'SI' element name, PDP takes the error exit.

6.2.2. FORTRAN Procedure

A FORTRAN procedure contains FORTRAN source language that is to be included in a compilation by use of the FORTRAN INCLUDE statement. The current version of the FORTRAN V programmers reference manual (a UNIVAC publication) should be consulted for rules concerning the files searched for a procedure at INCLUDE time. See the section on the INCLUDE statement. If no definition is found, the compiler gives an error indication.

The FORTRAN procedure has the form:

```
AA   PROC
      -
     (FORTRAN STATEMENTS)
     END
```

An entry will be made in the program-file FORTRAN procedure table for the label 'AA'. The END statement for FORTRAN procedures must begin in column two (2).

6.2.3. COBOL Procedure

A COBOL procedure contains COBOL source language that is to be included in a compilation by use of the COBOL INCLUDE and COPY verbs. Consult a current COBOL manual to determine compiler action upon encountering a COPY or an INCLUDE. If no definition is found, the compiler gives an error indicator.

The COBOL procedure has the form:

```
AA   PROC
      -
     (COBOL STATEMENTS)
      -
     END
```

An entry will be made in the program-file COBOL procedure table for the label 'AA'. 'END' must begin in column 2.

6.3. TEXT EDITOR (@ED)

The Text Editor is capable of manipulating images in Standard Data File (SDF) format files and/or elements of program files. It may be used to insert images, change images or portions of images, and to delete images. It can be used to make a copy of an SDF format file and insert it into a program file as an element, or conversely, to overwrite an SDF format (or unformatted) file with the contents of a symbolic element to produce a new SDF format file. The ED processor is conversational in nature and as such is mainly used in demand mode. However, there are occasional batch applications for which ED is appropriate. Details on the ED processor may be found in the Demand Terminal User's Manual for the UNIVAC 1108, an RECC publication.

6.4. The @ELT Processor

The @ELT control statement introduces an element into a particular program-file from the control stream. It may also be used to make corrections to a source element in a program-file. The element or the corrections follow the @ELT statement in the control stream.

The format of the @ELT statement is:

```
@ELT,OPTIONS SI,SO,SENTINEL
```

The options are:

A	Absolute Element
R	Relocatable Element
S	Symbolic Element
D	Data Element
I	Insert. Initial insertion of an element into a program file.
U	Update. Produce a new cycle of source language.
L	Produce a listing of the complete source element.

The options 'A', 'R', 'S', and 'D' identify the element types. Types 'S' and 'D' are both considered source language elements and may be corrected in the same manner (see section 4.6 'PROCESSOR CALL STATEMENTS'). A source language element in a program-file has the same format as the system data file. When an 'A', 'R', 'S', or 'D' option is not present the 'S' option is assumed.

The @ELT statement initiates the element processor which operates in one of two modes. It inserts new elements into the program-file from the control stream or updates an element already in the program-file.

The field 'SI' identifies the input element by file, element name, version, and cycle (when appropriate). Field 'SO', if specified, identifies the new output element.

When the 'I' option is specified, the element in the control stream is given the name specified in the 'SI' field and inserted into the program-file specified in the 'SI' field.

When the 'U' option is specified, the corrections in the control stream are applied to the element identified in the 'SI' field, and a new cycle of the source language is produced.

When the 'SO' field is present the corrections in the control stream are applied to 'SI', and a new source element is produced. It will be given the name specified in the 'SO' field and inserted into the program-file specified in 'SO' field.

The 'L' option will produce a complete listing of a source element. The 'L' option is not applicable for absolute or relocatable elements.

When the 'U' or 'I' option is not present and the 'SO' field is void, the 'L' option is assumed and 'SI' will be listed.

The 'data element' may contain control statements. Therefore, the data following the @ELT,D statement must be terminated with an @END statement with a sentinel exactly the same as found on the @ELT,D statement. The sentinel field need not be coded (blank sentinels). It is a six character field used to search out the proper @END sentinel. All images will be passed into the data element being created until an @END command is found with the same character string. The @ELT,D statement may be used to insert @RUN or @ADD control streams into a program-file as elements which may be called later by the @START or @ADD statement.

Element types 'A', 'R', and 'S' are terminated by the next control statement in the control stream. They need no corresponding @END command; therefore, no sentinel is necessary.

When an element is punched by a processor or by program utility routine (FURPUR), it is always preceded by a @ELT control statement. The 'FILENAME' on the punched @ELT is that of the file from which the element was punched. Such decks can simply become part of the input to subsequent runs. (The file name must be changed if the element is to be added to a file different

from the one from which it was punched.)

The automatic deletion rules apply to the insertion of elements by an @ELT control statement.

Examples

```
    @ELT,I  PF.E              . A new symbolic element
    ...                       . 'E' is inserted in the
    ...                       . Program-file 'PF'.
    SOURCE IMAGES
    ...
    ...
    @ELT,U  PF.E              . The corrections following
    ...                       . this statement are applied
    ...                       . to the element 'E' of
    CORRECTIONS IMAGES        . Program-file 'PF'.
    ...                       . The updated element 'E'
    ...                       . Replaces the old 'E' in
    ...                       . The program-file.
    @ELT  PF.E,PF.N           . The corrections following
    ...                       . This statement are
    ...                       . Applied to the element
    CORRECTION IMAGES         . 'E' to produce a new
    ...                       . Element 'N'.
    ...
    @ELT,L  PF.E              . Element 'E' will be listed.
    @ELT  PF.E                . Element 'E' will be listed.
    @ELT,L  PF.E              . The corrections following
    ...                       . This statement are applied
    ...                       . To the element 'E'.  The
    CORRECTION IMAGES         . New element will be
    ...                       . Listed, but a new element
    ...                       . Will not be produced.
    @ELT,IA  PF.E             . A new absolute element
    ...                       . 'E' is inserted in the
    ...                       . Program-file 'PF'.
```

```
ABSOLUTE IMAGES
...
...
@ELT,IR  PF.E                . A new relocatable element
...                          . 'E' is inserted in the
...                          . Program-file 'PF'.
RELOCATABLE IMAGES
...
...
@EOF
...
...
PREAMBLE IMAGES
...
...
@ELT,DI  PF.D,,X             . A new data element 'D'
...                          . is inserted in the
...                          . Program-file 'PF'.
DATA IMAGES
...
...
@END X
```

6.5. The @DATA Processor

The @DATA statement may be used to introduce standard format data files, found in the control stream, into the system for residence on a mass storage device. A primary use for this feature is to allow the user to build data files which are actually whole or parts of control streams. These files can then be called on by the @START statement to start an independent run, or by the @ADD statement for inclusion into the current run or a subsequent run. A data file correction feature is also available via the @DATA statement. The user can make a correction to an independent runstream and then @START it, or make corrections to a partial stream and then @ADD it to the run. The @DATA statement can of course simply be used as a convenient

means of generating and maintaining a user data file, rather than a control-stream type file.

The format of the file created as a result of the @DATA statement is the systems data file format.

The format of the @DATA control statement is as follows:

```
@DATA,OPTIONS   FILENAME1,FILENAME2,SENTINEL
```

The options field may contain the following characters:

I	Insert. Initial insertion of data into the file.
U	Update. Produce a new version of the data.
L	Produce a complete listing of the file.

The 'L' option will produce a complete listing of the file which will include sequential item numbers. These item numbers will be used when making corrections to the file. Corrections to the file are made in the same manner as corrections to a source language element. (See section on 'PROCESSOR CALL STATEMENTS'). If 'L' and 'FILENAME1' are the only information present in the @DATA statement, 'FILENAME1' will be listed.

When the 'I' option is present, the data following the @DATA statement is written to 'FILENAME1'.

When the 'U' option is present, the data following the @DATA statement is taken as corrections to 'FILENAME1' and a new F-cycle of 'FILENAME1' is produced. The next F-cycle of 'FILENAME1' must have been previously assigned by the user, as in:

```
@ASG,C     FILENAME1(+1), F2
```

If neither the 'U' or 'I' option is present, the data following the @DATA statement is taken as corrections to 'FILENAME1', and a new updated file ('FILENAME2') is created.

If neither the 'I' or 'U' option is present and the 'FILENAME2' field is void, the 'L' option is assumed and 'FILENAME1' is listed. No new file will be generated.

The data following the @DATA statement is terminated with an @END

statement with a matching sentinel. As in the @ELT,D statement a search is made for the appropriate @END with all images in between placed in the data file.

Any control statements (except @FIN) appearing between the @DATA statement and the end of data sentinel @END are treated as data by the system. This allows control streams to be entered as files and called later for execution.

Demand users should note that if a @DATA statement is rejected for any reason, an @END statement is still required. Otherwise, additional control statements will be considered as data and ignored.

Examples:

This statement will generate a new file 'X' containing the data following the statement.

```
@DATA,I  X
...
...
data images
...
...
@END
```

This statement will apply the corrections to file 'X' and create a new file 'Y'.

```
@DATA  X,Y
...
correction images
...
...
@END
```

This statement will list file 'Y'.

```
@DATA,L  Y
@END
```

This statement will list file 'Y'.

```
@DATA  Y
@END
```

This statement will apply the corrections following the statement to file 'X'. The new file will be listed, but a new file will not be generated.

```
@DATA,L  X
...
...
correction images
...
...
@END
```

This statement will generate a new file 'A' containing the data following the statement. File 'A' will also be listed.

```
@DATA,IL  A
...
...
data images
...
...
@END
```

This statement will apply the corrections following the statement to file 'X' and create a new F-cycle of file 'X'.

```
@DATA,U  X
...
...
correction images
...
...
@END
```

This is an example of nested @DATA statements. The sentinel field must be coded on one set of @DATA/@END statements.

```
@DATA, I  RUN
@RUN
@ASG,T    FIL,F
@DATA,I     FIL,,SENT
@PACK
@PREP
@PRT,T
@END      SENT
@COPY,P   PROGFIL
@ADD      FIL.
@END
```

6.6. The @END Statement

The @END control statement marks the end of the data that follows a @DATA or @ELT,D statement. The format of the @END statement is:

```
@END    SENTINEL
```

This statement cannot be continued on a second line. The sentinel field is optional. It is coded exactly the same as the corresponding field on a @DATA or @ELT,D statement when being used to bracket images of the data.

6.7. The @LF Processor

Users need a method of obtaining a list of their catalogued mass storage files and of determining certain characteristics of each file. The List Files Processor LF has been designed to do this efficiently, both in terms of machine time used and printout produced. User convenience was the prime consideration, and resulted in such features as (1) specially formatted teletype output, (2) a wide range of listing options so that unwanted information need not be printed, (3) the ability to handle all files belonging to a user or only files specified by him, and (4) processor initiation and termination messages to inform the user of the status of his run while directory information is being processed.

6.7.1. The @LF Processor Call Statement

The user directs the action of LF via a processor call statement of the form:

```
@LF,OPTIONS    FILENAME1,..., FILENAMEn
```

'FILENAME1,...,FILENAMEn' specify in standard FILENAME format, a list of catalogued files to be processed by LF. If no such specifications are given, then LF will process all files belonging to the user.

The options fall into two categories: listing options and general options. The listing options determine what information is to be printed for each file processed. In general, each listing option causes the printing of one or more codes (an abbreviation for the type of information being given), followed by a hyphen and then the actual information. For example, the S option causes the code SZ (an abbreviation for size) to be printed, followed by a hyphen and the actual size of the file (an integer equal to the number of tracks the file currently occupies). A summary of the listing options follows.

LISTING OPTION	CODE(S)	MEANING
A	F2	Allocation on Fastrand II
	F4	Allocation on FH432
	F8	Allocation on FH880
	F17	Allocation on FH1782 The number of tracks the file is occupying on each equipment type. If a particular equipment type is not allocated on, then its code will not be printed.
B	TFW	Time of First Write since Backup The date and time that the file was first written into since the last time the computer center copied mass storage to tape. NONE is printed if the file has not been written into, indicating that the computer center's tape backup has a current copy of the file on it.
	BU	Backup Date and Time The date and time that the file was last copied to tape by the computer center.
C	CA	Current Assigns The number of runs that currently have the file assigned to them.
H	HGA	Highest Granule Assigned The number of the highest granule, relative to the beginning of the file, that is assigned to it.
	HTR	Highest Track Referenced The number of the highest track referenced, relative to the beginning of the file.

N	NA -	Number of Assigns The total number of times the file has been assigned.
O	(none)	Options The options used on the initial @ASG or @CAT for the file are printed to the left of the filename. These include P(public), R(read-only), W(write-only), G(guarded-no backup allowed), and V(unload inhibit). X is printed if the file is currently exclusively assigned to some run.
P	(none)	Parameter The facilities field from the initial @ASG or @CAT of the file is printed to the right of the filename. For mass storage files, this includes equipment type, initial reserve, granularity, and maximum granules. This is the information used to detect files catalogued with an illegal equipment type.
R	(none)	Reference Number The reference number under which the file was catalogued.
S	SZ	Size The number of tracks occupied by the file. This is the information used to detect files that are too large.
T	CAT	Time of Cataloguing Date and time are printed.
	LR	Time of Last Reference Date and time the file was last assigned are printed. This is the information used to detect expired files.
U	(none)	User Name The user name under which the file was catalogued.

If no listing options are given, then only the filename is printed. General options direct the overall operation of the processor, and include the following.

GENERAL OPTION	MEANING
L	Detailed Listing Specifying the L option is equivalent to specifying all the listing options.
D	Delete Undesirable Files Files of size 0, of size greater than the current legal maximum, and with improper equipment type are deleted. Only the deleted files are listed. The listing options specify the information to print concerning each file deleted.

E	Delete Expired Files Files whose time of last reference is prior to the current date minus the expiration period are deleted.
F	Sort The listing produced is sorted by qualifier, file, and F-cycle.
W	Wide Page Output is formatted to fit a 132 character line (batch or DCT 500). If W is not specified, output is formatted to fit a 72 character line (teletype or equivalent).

6.7.2. Functional Aspects of @LF

The @LF processor never assigns a user's file to the run; hence an @LF statement never alters the last reference time for any file. Thus, a periodic @LF of files is not sufficient, in itself, to protect files from expiring.

The temporary file TDIR$ is assigned to the run for holding directory information. If @LF terminates normally, this file is @FREE'd. If no specifications were given, @LF requests that the executive copy the directory into the file TDIR$, causing a noticeable delay between the LF sign-on and the first line of information. If specifications were given, a check is made to determine if the specified files are assigned to the run. If so, the copy of the entire directory is avoided and delay is eliminated. Thus a user may speed up action of @LF by assigning all files named in @LF specifications prior to entering the @LF processor call.

Whenever an @LF call has been processed, the message END LF and the total number of tracks occupied by the files processed will be printed. @LF will not terminate; rather, it will look to see if the next control card is an @LF call. If so, it will be processed immediately using directory information already retrieved. Thus, the delay required to copy the directory into TDIR$ need only occur once for many @LF calls.

For the demand user, this means @LF may be called once to list all files (causing a directory copy), then again to list detailed information about particular files. The additional @LF calls will be processed without a delay for copying the directory.

This also means that all information listed by a series of @LF calls will reflect the directory as it was for the first @LF call. If a demand user wishes to monitor the status of the directory he must enter a control statement other than @LF between @LF calls. This would be useful, for example, to monitor the current assigns for a file referenced by a @START run. When the current assigns increases, the run has started; when it decreases, the run has finished.

Certain error conditions cannot be handled by @LF. In this case, the message ERROR IN PROCESSOR is printed. These conditions are limited, for the most part, to hardware device I/O errors and improper directory structure. The ERROR IN PROCESSOR message may indicate that the integrity of the master file directory is in peril, and it would be wise for the user to transfer recent changes to his mass storage files to another media such as punched cards, punched paper tape, or magnetic tape.

6.7.3. Examples of the @LF Statement

```
@LF
```

The qualifier, file name, and F-cycle for each file catalogued by the user (under any reference number) are listed.

```
@LF,L
```

All catalogued file information pertaining to the user is printed. It might be instructive to compare this output to the output of a @PRT statement.

```
@LF,OPA  PROGFILE,ØUR*DATAFILE.,X(3).
```

The cataloguing options, cataloguing parameters, and allocation summary are printed for the most recent F-cycles of files user-name *PROGFILE and OUR*DATAFILE, and for F-cycle 3 of file user-name*X. Note that if any of the FILENAME subfields are specified in addition to the FILE subfield, the field must be terminated with a period.

```
@LF,PAT
```

The cataloguing parameters, allocation summary, and dates and times of cataloguing and of last reference are printed for each of the user's

files. This information is sufficient to determine if a file is subject to deletion by the computer center. Note that a file of zero size will not have any allocation summary, since it does not actually occupy any equipment type.

6.8. The LIST Processor

The LIST processor is designed to produce a readable edited listing of all types of elements. In connection with the notes given below the listings are self-explanatory. If, because of some error, an element is too badly formed to be listed, features are available for dumping the element in octal.

6.8.1. The Processor Call Card

The general format of the processor call is:

```
@LIST,options   E1,...,En
```

Where E1,...,En are element names in full element notation. The files mentioned in the element names may be either catalogued or assigned to the run. If the element name Em (m>1) has a leading '.' then the file used will be the file used for the previous specification. The element cycle subfield is ignored.

The following options determine the types of elements listed:

A - List absolute elements (type 6),

R - List relocatable elements (type 5),

S - List symbolic elements (types 1,2,3, and 4).

At most one of these options may be coded. If none of these options are coded, then the 'S'-option will be assumed.

If the 'O'-option is coded in addition then the elements will be dumped in octal with no attempt at editing.

6.8.2. Notes on the Printed Output

6.8.2.1. Symbolic Elements

1. Every SDF image in the element, including control images, will be printed along with the image length and the relative word address of the image.

2. Source images which belong to the most recent symbolic cycle will be numbered. The cycle information for all source images will be printed.

3. If the symbolic element is an Assembler, COBOL, or FORTRAN procedure, the appropriate procedure name table will be printed.

6.8.2.2. Relocatable Elements

1. Each text word will be printed as twelve octal digits. The j-field (bits 29-26), a-field (bits 25-22), x-field (bits 21-18), and hi-fields (bits 17 and 16) will be printed below the text word.

2. The following abbreviations are used when the relocation information is printed:

   LC - Location Counter

   XR - External Reference

   LH - Left Half (bits 35-18)

   RH - Right Half (bits 17-0)

   LA - Left Address (bits 33-18)

   RA - Right Address (bits 15-0)

6.8.2.3. Absolute Elements

1. See note 1 for relocatable elements.

2. The following abbreviations are used when printing the relocation information for relocatable segments:

   L - Left half relocated

   R - Right half relocated

6.9. The TSTCAT Processor

The TSTCAT processor is used to test for the existence of a catalogued file, recreate it if necessary, and assign said file to the user's run. Normally, RECC attempts to guarantee the presence of files such that the TSTCAT processor need not be used. However, a user may be required to maintain a file himself as a G-option file if, for instance, the size of the file exceeds the current maximum allowable size. In this case, TSTCAT is a valuable tool.

When TSTCAT is called, it will attempt an assign of the specified filename from the directory. If the attempt is successful, TSTCAT exits normally. If the assign is rejected because the file is not in the directory, the options and specifications control TSTCAT processing.

The format of the processor call card is:

```
@TSTCAT,OPTIONS      FILENAME.,TAPE/REEL,MOVE,MAXT
```

The available options are:

- C - Do not terminate run if there is a TSTCAT or FURPUR error
- E - Create an entry point table for the file (i.e., issue a @PREP command)
- F - Create the file with @COPY,F instead of @COPIN
- G - Create the file with @COPY,G instead of @COPIN
- I - Do not free the tape if it had to be assigned
- M - Medium density tape (556 bpi)
- P - Create public file if recataloguing is done
- R - Create read-only file if recataloguing is done
- X - Assign file with exclusive use

The specifications are interpreted as follows:

FILENAME - Catalogued mass storage filename in standard notation.

TAPE - Tape filename to be used if recataloguing is necessary.

REEL - Reel number of TAPE.

MOVE - Number of end-of-files to move over before file is reached.

MAXT - Maximum size of file, in tracks.

MOVE and MAXT are optional; if they are present, TAPE/REEL must be present. If MOVE is omitted, no move is performed; if MAXT is omitted,

128 is assumed. High density and @COPIN are assumed unless overridden with options.

If the file can be assigned from the directory (@ASG,A successful), T3 of the condition word is set to 0000, and TSTCAT exits. If the file is not in the directory, T3 of the condition word is set to 0001. If only FILENAME is specified on the processor call card, an exit is taken. The user may use conditional control statements to control his run, and take any action he desires upon detecting the absence of the file.

If specifications other than FILENAME are given, TSTCAT will build a partial runstream in the file TSTCAT$ADDFL and @ADD it to the run. The added runstream will assign TAPE, catalogue FILENAME with a G option, and load the text of FILENAME from TAPE as specified by the options.

7. THE DIAGNOSTIC SYSTEM

7.1. The @PMD Statement

7.1.1. General

A POST-MORTEM DUMP executive control statement may be used to dump core memory following the execution of a task. Dumps may be made of overlay segments, elements, or specified parts of elements, as long as they were currently in core at the time the routine terminated. Several options are available for output formatting and for selecting the core areas to be dumped.

The general form of the control statement is:

```
@PMD,OPTIONS        SPECIFICATIONS
```

If no information was saved by the system when the previous execution terminated, no dumps are possible. This condition may be caused by a 'Z' option given to the COLLECTOR when the program was constructed, by a misplaced @PMD card, or by certain rare error conditions. In the event that no dump is available, a message is produced.

The @PMD statements must follow the @XQT statement of the program that has terminated in order to be honored. Only pure data, @EOF's, and the conditional statements: @SETC, @JUMP, and @TEST may intervene. An example follows:

```
NO.       STATEMENT
----    ------------------
 1        @XQT   PROGX
 2        DATA
 :          :
 :          :
10        DATA
11        @TEST       TE/6/S3
12        @JUMP       3
13        @SETC       6/S4
14        @PMD        ELEMENT-1, ELEMENT-2
15        @XQT        PROGY
 :          :
 :          :
```

If PROGX terminates before processing all of the data statements that follow the @XQT and, if S3 of the condition word has a value of 6, then S4 of the condition word will be set to 6 and statement numbers 14, 15 will be honored for processing. When statement 16 is encountered, the run will be terminated if it is not demand.

7.1.2. Options

Options are selected through use of option letters punched into the @PMD card. The options fall into the following classes: (1) General, (2) Special, (3) Options with Specials, and (4) Blank.

7.1.2.1. General Options

The general options may be used with any others in a @PMD statement. They are:

(1) 'E' option" if the letter E is placed in the options field, the @PMD statement will be processed only when the previous routine terminated in error.

(2) 'C' option: the'C' option will cause a dump of the words that were changed during the execution of the allocated program for the area of core prescribed by the specifications portion of the @PMD statement.

(3) 'B' option: after processing the rest of the @PMD statement, this option will cause an octal dump of all of blank common's storage.

(4) 'P' option: the letter 'P' used in conjunction with any of the other options known to PMD will cause an octal dump of the PCT block(s) used by the run to be printed preceding the dump of the program. The blocks are dumped in octal format. The segment load tables (if any exist) are also dumped in octal format if the 'P' option is specified.

7.1.2.2. Special Options

Only one special option should be used on a single @PMD statement. If more than one special option is used, the special 'A' option is assumed. All special options require the specifications field described below. If

no special option is supplied, the blank option rules will be applied. The specifications field for the special options takes the form of a list of element or segment names:

NAME 1, NAME 2, NAME 3, (ETC.)

Each entry will be dumped in octal format and in order of allocation. If the specification field is blank, all elements in memory at termination of the previous routine will be dumped. These special options are:

(1) 'A' option: an 'A' option will produce a dump of all memory specified in each element or segment named in the specification list.

(2) 'D' option: a 'D' option will produce a dump of the D-bank portion of each element or segment named in the specification list.

(3) 'I' option: an 'I' option will produce a dump of all I-bank portions of each element or segment named in the specification list.

7.1.2.3. Options Used with Special Options

(4) 'X' option: When used in conjunction with the 'A', 'I', or 'D' options, the 'X' option has an except effect. All active elements will be dumped except those named in the specification list, and those belonging to the segments named in the specification list.

(5) 'L' option: When used with the 'A', 'I', or 'D' options, the L option, present, causes a dump to be taken of any active elements from the system library. The 'L' option when used alone will cause the active library elements to be dumped.

7.1.2.4. The 'Blank' Option

If no special options are named on the @PMD card, the specification field must follow the form:

NAME, START, LENGTH, FORMAT

This option allows the user to dump information under a specific

format without outputting excessive amounts of unnecessary material.

The 'NAME' field is that of an element and must be present.
The 'START' field must of the form:

N/M

Where 'M' represents the location counter of the element to be dumped, and 'N' represents an address, relative to the beginning of 'M', at which dumping should begin. If 'M', or 'N' is omitted, a zero is assumed to be its respective value.

The 'LENGTH' field must be the number of words to be dumped. If omitted, the length will be assumed to be all of location counter 'M' of the specified element.

The 'FORMAT' field may contain either a one-letter code for a system defined format, or a user defined format in FORTRAN notation. The system defined formats are:

F	(8 F 14.8)	Fixed Decimal
E	(8 E 14.8)	Floating Decimal
I	(8 I 14)	Integer
A	(16 A 6)	Alphanumeric
O	(8 O 14)	Octal
S	(4 S 30)	Instruction
D	(4 D 26.18)	Double Precision Floating Point

Standard 'D', 'S', and user defined formats are not applicable for changed words dumps; for all other cases the user may apply his own FORTRAN type formats (enclosed in parentheses) or use the system defined formats previously mentioned.

7.1.3. Examples

@PMD

Results in an octal dump of all active (allocated in core) segments of a users program. No blank common will be dumped.

@PMD,EAXL ELEMENT-NAME-1,ELEMENT-NAME-2

Results in an octal dump of all active elements except ELEMENT-NAME-1, ELEMENT-NAME-2, and system library elements on an error terminations.

@PMD,BDI SEGMENT-NAME

Results in an octal dump of SEGMENT-NAME (if active) and blank common area of core storage.

@PMD,EBCD ELEMENT-NAME

Results in an octal dump of changed words in DBANK of ELEMENT-NAME (if active) and blank common if the program terminated in error.

@PMD ALPHA,100/3,50,A

Results in a 50-word alphanumeric format dump of element ALPHA (if active under control of location counter 3 beginning with relative address 100) of location counter 3.

@PMB,B . DUMP ALL OF PROGRAM INCLUDING BLANK COMMON.

8. UTILITY ROUTINES

8.1. Conversion Aids

To aid installations in the transition from other computers to the UNIVAC 1108, (Exec 8), a set of conversion routines are incorporated into the 1108 Executive System. Those conversion routines desired by an installation will be included in the systems library, and defined as systems processors at system set up time. The routines may be called as desired by use of the executive processor call statement.

The following conversion aid routines are provided:

8.1.1. UNIVAC 1108 (EXEC II) to UNIVAC 1108 (EXEC 8)

This processor will convert magnetic tapes created by the Exec 2 complex utility routine (CUR) to magnetic tapes acceptable as input to an Exec 8 program file. The processor will accept Exec 2 symbolic elements, COBOL library elements, and procedure elements and convert them to Exec 8 symbolic elements, COBOL procedure elements, and assembler procedure elements, respectively. All other Exec 2 element types will be ignored.

The processor resides in the systems library and is initiated by the following processor call command

```
@CON78,OPTIONS  FILE1.,FILE2.
```

The available options which indicate element type are as follows:

S	Symbolic elements
C	COBOL library elements
P	Procedure elements

If none of the options is specified, all elements of type S, C, and P are converted.

'FILE1' is the input tape file of Exec2 elements and 'FILE2' is the output assigned to the run with the 'ASG' control command. One call to @CON78 converts one file, two calls converts two files, etc. It should be noted that a period is required to define the file name. The files are never rewound and end of file marks are written after each file converted.

COBOL procedures are handled somewhat distinctly. Each procedure is surrounded by a proc line and an end line. The label on the proc line is that

of the element name and it is not externally defined. Therefore, in order to use the procedure in a compilation it will be necessary to PDP it.

8.1.2. LIFT (FORTRAN II to FORTRAN V Translator)

LIFT is a source language translator which accepts a FORTRAN II source language program as input, performs a translation, and prepares a source language program acceptable to the FORTRAN V Compiler. There is a need for translation since FORTRAN II is not a proper subset of FORTRAN V. LIFT itself, written in FORTRAN V, is fully integrated with the Executive System.

There are nine areas of incompatibility between FORTRAN II and FORTRAN V, and the basic purpose of LIFT is to generate FORTRAN V Source Statements which replace the unacceptable FORTRAN II Statements.

1. The "F" Card
2. Functions
3. Boolean Statements
4. Double-Precision and Complex Statements
5. COMMON Statements
6. Arithmetic Statement Functions
7. Dimension Statements
8. Hollerith Literals
9. Implicit Multiplication

There are also five types of FORTRAN II statements that, although acceptable to the FORTRAN V processor, are converted to their FORTRAN V equivalents. LIFT offers two features that ease the transfer between computers: the ASSIGN and REPLACE card options. The ASSIGN card allows a temporary change to be made to the I/O Assignment Table, and the REPLACE card allows the user to have every occurrence of a variable name replaced with another variable. The standard output produced by LIFT consists of a listing of the FORTRAN II program, an annotated list of the translated program, and a symbolic program element suitable for use as input to any FORTRAN V Compiler.

8.2. The TD8 Routine

The function of TD8 is to produce dumps of tapes and/or drum areas (FH-type or FASTRAND).

8.2.1. Execution

TD8 is called by:

```
@XQT  GT*LIB.TD8
```

8.2.2. Data Card

An optional data card may follow the @XQT control statement. This card contains information about the number of blocks (or words, or sectors) to be dumped and a description of the editing format. The fields on the card in detail:

Column	Contents
1 - 6	Number of records to be dumped.
7	A- (for Alpha) or O- (for octal) format.
10 - 12	Number of data words to be edited on one print line (≤ 8 for O,≤21 for A).
13 - 18	Number of blocks or words or sectors to be skipped before dumping (blocks for tape, words for FH-drum, sectors for FASTRAND).
19 - 30	Filename for facility.
31 - 42	Element name } only if an element has to be located in an EXEC 8 element file.
43 - 54	Version name }

The term 'records' for the first field represents blocks for tape, 256 words for FH-drum and 10 sectors for FASTRAND. Any one of the above fields may be omitted if all the following fields are omitted too. For the first five fields the program contains standard definitions. If some or all of these fields are blank, the standard values will be used. They are:

a) Number records = 100 (=R).
b) Format = 0 (octal) (=F).
c) Number Words per line = 8 (=L).
d) Skip = 0 (=S).
e) Filename = DUMPFACIL$ (=N).

8.2.3 Results

The TD8 will either read R records or read until it encounters an

end-of-file condition. The data blocks on tape should not be more than 4096 words in length. Each record is edited according to the format specifications (F and L). The listing of a record is preceded by one header line which contains length and number.

Before the read and dump process starts, the skip parameter S is checked and if the facility type is tape, S blocks are skipped; for drum, the relative address for the first read will be S (this is for both FH-drum and Fastrand).

The name, found in Columns 19-30, or if none is specified, the name DUMPFACIL$ is used in the I/O packet. Therefore, the runstream must contain an @ASG and/or @USE control statement, prior to the @XQT GT*LIB.TD8 card.

8.2.4. Example

The following runstream will dump 25 blocks from tape file DATATAPE, reel U325, in alphabetic format. There will be 14 six-character words per line in the dump listing.

```
@RUN      TDUMP,15T3905,  TRACEY-T-M
@PWRD TT59X5
@ASG,T  DATATAPE,T,U325N
@USE    DUMPFACIL$,DATATAPE
@XQT    GT*LIB.TD8
    25A  14
```

9. SAMPLE DECK SETUPS

The following sections will cover some of the most common examples of program deck setups. It should be noted that these examples are general in nature and do not cover all possible variations. These examples are batch oriented as demand examples are given in the RECC publication, Demand Terminal User's Manual for the UNIVAC 1108.

9.1. Compile Only

```
@RUN TEXNDA,15M800099,DOE-J
@PWRD CCJKLM
@COB,SBE
.....
COBOL Source Language
.....
@FIN
```

In this example, the COBOL source language program is compiled and the resulting relocatable element put into the run-temporary file. The specifications on the @RUN statement refer to the RUNID, REFERENCE-NUMBER, and USER-NAME, respectively, reading from left to right.

9.2. Compile and Execute

```
@RUN  TEST1,21S800001,DOE-T,1,50
@PWRD ME1557
@ALG,IS  TSUAC
ALGOL Source Language
....
@MAP,S
@XQT
....
data cards
....
@FIN
```

In the above example, the ALGOL source language is compiled, and the resulting relocatable element, TSUAC is @MAP'ed, and the absolute executed.

9.3. Compile and Execute Main Program With Two Subroutines

```
@RUN TEST2,25A4137,DOE-S
@PWRD JC1M9Q
@FOR,IS MAIN
....
FORTRAN Source Language (main program)
....
        END
@FOR,IS SUBR
        SUBROUTINE SUBR
....
FORTRAN Source Language (subroutine SUBR)
....
        END
@FOR,IS DIVIDE
        SUBROUTINE DIVIDE
....
FORTRAN Source Language (subroutine DIVIDE)
....
        END
@MAP,S
@XQT
....
data cards
....
@FIN
```

The above example illustrates the compilation of a main program and two subroutines, which will be allocated together and executed.

9.4. Compile and Catalogue Original Program

```
@RUN CATEST,38H100250,DOE-D
@PWRD NE1108
```

```
@DELETE,C  PROGFILE/READK/WRITEK.
@ASG,U PROGFILE/READK/WRITEK,F2
@FOR,IS PROGFILE.MAIN
....
FORTRAN Source language (main program)
....
      END
@FOR,IS PROGFILE.SUBR
      SUBROUTINE SUBR
....
FORTRAN Source language
....
      END
@FOR,IS PROGFILE.DIVIDE
      SUBROUTINE DIVIDE
....
FORTRAN Source language
....
      END
@MAP,IS    ,PROGFILE.MINT2
IN         PROGFILE.
@FIN
```

The @DELETE insures that the file PROGFILE is not already catalogued.

The assign (@ASG) statement is used to name an external file, set up its I/O requirements, and catalogue the file for future reference. The "U" option on the ASG card specifies that the file is to be catalogued regardless of the manner of termination of the run. "PROGFILE" is the name of the file to be assigned to the run. "READK" and "WRITEK" are specification fields which prevent reading and writing of the user's file by other users. To gain access to the file, the appropriate keys must be specified at assign time or the assignment will not be made. "F2" indicates that the file will be located on Fastrand.

The processor call statement (@FOR) specifies that a source language element will be introduced into the file "PROGFILE" from the control stream and also given to the processor for compilation. The "I" option on the

@FOR card directs this introduction of source language from the runstream. The @MAP and the IN place an absolute element ready for execution in PROGFILE.

9.5. Test Corrections to Existing Program and Execute

The existing program will not be altered.

```
@RUN       CATST1,38H100250,DOE-D,20,300
@PWRD NE1108
@ASG,A     PROGFILE/READK
@COPY,R    PROGFILE.
@FOR,WS    PROGFILE.MAIN,MAIN
-5
       CALL DIVIDE
-10,10
       C = SQRT (A*A+B*B)
-31,27
       A = MAX (A, B, C)
       CALL SUBR
@MAP,IS
@XQT
...
data cards
...
@FIN
```

On the @RUN statement the "20" in the specifications field is programmer estimated run-time. If the run exceeds this time, the run will be terminated. The "300" in the last specification field means that a maximum of 300 pages of output is expected from this run. If exceeded, the run will be terminated automatically.

The assign (@ASG) statement is used to name an external file, "PROGFILE," which contains the program source language for this run. Reference 9.4., where this file was catalogued. The write key is not given to prevent accidental writing into the file.

The @COPY,R brings the relocatables from PROGFILE into TPF$. This is necessary if any of the elements are not going to be recompiled.

The @FOR compiles the sumbolic PROGFILE.MAIN, with corrections as indicated. (See section 4.6.3 for a discussion of correction statements) The resulting relocatable is TPF$.MAIN which deletes the relocatable MAIN copied from PROGFILE.

The @MAP collects the relocatables in TPF$ into the absolute TPF$.NAME$. These are the new MAIN and the original SUBR and DIVIDE.

9.6. Update Existing Program and Execute

```
@RUN CATST1,38H100250,DOE-D,20,300
@PWRD NE1108
@ASG,A PROGFILE/READK/WRITEK
@FOR,UWS PROGFILE.MAIN
....
Correction statements
....
@MAP,IS  ,PROGFILE.MINT2
IN PROGFILE.
@PACK     PROGFILE
@PRT,T    PROGFILE.
@XQT PROGFILE.MINT2
....
data cards
....
@FIN
```

In the above example, corrections will be made to FORTRAN element "MAIN" before compilation. The correction statements must immediately follow the processor call statement. The "U" option on the @FOR statement specifies that an updated source language element will be produced by applying the corrections to the input source language. The "W" option specifies that all correction statements will be listed.

Subroutines "SUBR" and "DIVIDE" are also part of this program (see previous two examples). Since there are no modifications to these routines,

they are not recompiled.

The main element "MAIN" with subroutines "SUBR" and "DIVIDE" will be collected together, forming an absolute element, "MINT2," which will also be placed in the file, "PROGFILE." "PROGFILE" is @PACK'ed to minimize the size of the file, and @PRT,T'ed for later reference. The absolute program is to be executed; data cards follow the @XQT statement.

9.7. Execute Existing Programs Using Cataloged Data Files

```
@RUN MINT,38H1002050,DOE-D
@PWRD NE1108
@ASG,A PROGFILE/READK
@ASG,AX DOE*MASTER-FLT
@USE MASTER,DOE*MASTER-FLT
@ASG,T  TEMP,F/3
@XQT PROGFILE.MINTI
....
data cards
....
@FREE DOE*MASTER-FLT
@ASG,AX MINT
@USE OLD,MINT
@ASG,CR MINT(+1),F2
@USE NEW,MINT(+1)
@XQT PROGFILE.MINT2
....
data cards
@FIN
```

In this example, two programs, "MINTI" and "MINT2", are to be executed in respective order. The programs are currently contained as absolute elements in file "PROGFILE".

The program,"MINTI", reads data cards and a file, "MASTER". For this run, a file created under the user name of "DOE" is to be used as "MASTER". A temporary file, "TEMP", is created.

The program "MINT2", updates the file, "MINT", referencing the current cycle of the old file as "OLD" and the updated cycle of the file as "NEW".

Note that the file, "DOE*MASTER-FLT", is not required by the run after "MINTI" is terminated. The @FREE statement releases the file so that another run might gain exclusive access to the file during the time that "MINT2" is being executed.

9.8. Compile Program and Store It on Tape

```
@RUN CRTAP,30N991108,DOE-S
@PWRD SMPRIE
@ASG,T  PRGTAP,T
@FOR,IS MAIN
....
FORTRAN Source language
....
        END
@FOR,IS SUBR
        SUBROUTINE SUBR
....
FORTRAN Source language
....
        END
@MAP,IS ,PRGXQT
@COPOUT        ,PRGTAP
@SAVE          PRGTAP
@FIN
```

The assign (@ASG) statement is used to inform the operator that a scratch tape is needed for this run.

The main program and subroutine will be inserted in the temporary program file (TPF$). The @MAP statement creates an absolute element for this program. Note that this will save the allocation time when the job is later executed (in another run).

The @COPOUT statement transfers all elements in the temporary program

file (TPF$) to the tape (PRGTAP).

The @SAVE requests the operator to save the tape and return its reel number.

9.9. Execute Program Stored on Tape

```
@RUN USTAP, 30N991108,DOE-S
@PWRD SMPRIE
@ASG,T PRGTAP,T,U305N
@COPIN PRGTAP
@FREE PRGTAP
@XQT PRGXQT
....
data cards
....
@FIN
```

The assign (@ASG) statement informs the operator to mount reel number U305 without a write ring ("N"). This tape was created by the previous example. The @COPIN statement transfers the program from tape to the temporary program file (TPF$). Note that in the previous example an absolute element, named "PRGXQT" was created. Hence, the allocation time will be saved each time this program is executed. The @FREE statement releases the tape servo so that another run may use it.

9.10. Create Multiple Print Output Copies

```
@RUN  PRODCT, 55K9909, JONES
@PWRD JCSSN1
@DELETE,C    PRINT
@ASG,URG     PRINT,F2
@MSG  EXPECT 5 PRINT FILES
@BRKPT  PRINT$/PRINT
@FØR,IS   PROG
   (FORTRAN statements)
@MAP,S
@XQT
     (data)
```

```
@BRKPT      PRINT$
@FREE       PRINT
@SYM,U      PRINT
@SYM,U      PRINT
@SYM,U      PRINT
@FIN              (pink card)
```

The @DELETE insures that no old copies of the file remain on Fastrand. The @ASG assigns the file PRINT which will contain the output from the run to be printed three times. PRINT is to be catalogued regardless of type of run termination, read-only, and is not to be saved on backup tapes at the end of day. The @MSG statement informs the operator that five output files are to be created, all to be returned to the same bin. The 5 in the @MSG statement specifies the number of copies desired plus 2 (two extra print files for the printing of initial and final control statements.) The first @BRKPT statement closes the initial print file and diverts further print to the file PRINT. The statements following the first @BRKPT may be any statements that generate the desired output. The second @BRKPT closes to the file PRINT and opens the final print file for the run. The @FREE statement causes final cataloguing action on PRINT, and makes it available to the system. Each @SYM statement directs the queing of the file PRINT to be printed (there should be one for each copy desired). After printing, PRINT will remain catalogued until it is deleted by the user or a full reboot is done.

If the run errors while creating PRINT, only two print files will be printed, the initial file and a file containing accounting information for the run. The user must submit a run to @SYM the file PRINT to receive the remaining output from the run. This may also be done from a demand terminal.

9.11 Divert Print Output to Tape

```
@RUN      PRODCT,55K9903,DOE-J
@PWRD JCSSN1
@ASG,T    PRINT,T
```

```
@MSG   EXPECT 2 PRINT FILES
@SAVE  PRINT
@BRKPT PRINT$/PRINT
   (Control statements, source language, and/or data needed to
    produce print)
@BRKPT  PRINT$
@FIN    (pink card)
```

9.12. Print Output Previously Diverted to Tape

```
@RUN   PRINT,55K9903,DOE-J
@PWRD JCSSN1
@ASG,T   PRINT,T,U1014N
@DELETE,C  PF
@CAT,G  PF,F2///1000
@COPY   PRINT,PF
@SYM  PF
@MSG  EXPECT 2  PRINT FILES
@FIN
```

The file PF will be decatalogued following printing.

9.13. Run Two Runs in Sequence

```
@RUN   RUN1,51J9308,SMITH  (note no S option)
@PWRD AB1234
@MSG   FIRST RUN OF 2 IN SEQUENCE
@XQT      X.UPDATE
    (data cards)
@FIN       (pink card)
@ BIN      (note fixed format: 1 space between "@" and "BIN")
@RUN,/S   RUN2,51J9308,SMITH  (note S option specified)
@PWRD AB1234
@MSG  SECOND RUN OF 2 IN SEQUENCE
@FØR,IS   LIST
  (FORTRAN statements)
@MAP,S
@XQT
@ADD      X.DATA
@FIN            (pink card)
```

The I/O clerk will add one standard bin card to the front of both decks. The decks must not be separated. Output for both runs will be returned to the same bin.

APPENDICES

A. Character Codes for the U 1108

B. Diagnostic Messages

C. Standard Tape Translation (BCD-Fielddata)

APPENDIX A

U 1108 CARD CODES

Char In Machine	Fieldata (Octal)	029 Mode			026 Mode			Key On Teletype	Char Printed On TTY
		Holes	Key On 029	Key On 026	Holes	Key On 029	Key On 026		
@ (1)	00	7-8	≥	None(3)	7-8	≥	None(3)	@	@
[	01	12-4-8	[	)	12-5-8	(	None(3)	[(shift K)	[
]	02	0-6-8	]	None(3)	11-5-8	)	None(3)	](shift M)	]
#	03	3-8	#	=	12-7-8	←	None(3)	#	#
∆	04	11-7-8	≤	None(3)	11-7-8	≤	None(3)	↑	↑
(Blank)	05	None	(Space)	(Space)	None	(Space)	(Space)	(Space)	(Space)
A	06	12-1	A	A	12-1	A	A	A	A
B	07	12-2	B	B	12-2	B	B	B	B
C	10	12-3	C	C	12-3	C	C	C	C
D	11	12-4	D	D	12-4	D	D	D	D
E	12	12-5	E	E	12-5	E	E	E	E
F	13	12-6	F	F	12-6	F	F	F	F
G	14	12-7	G	G	12-7	G	G	G	G
H	15	12-8	H	H	12-8	H	H	H	H
I	16	12-9	I	I	12-9	I	I	I	I
J	17	11-1	J	J	11-1	J	J	J	J
K	20	11-2	K	K	11-2	K	K	K	K
L	21	11-3	L	L	11-3	L	L	L	L
M	22	11-4	M	M	11-4	M	M	M	M
N	23	11-5	N	N	11-5	N	N	N	N
O	24	11-6	O	O	11-6	O	O	O	O
P	25	11-7	P	P	11-7	P	P	P	P
Q	26	11-8	Q	Q	11-8	Q	Q	Q	Q
R	27	11-9	R	R	11-9	R	R	R	R
S	30	0-2	S	S	0-2	S	S	S	S
T	31	0-3	T	T	0-3	T	T	T	T
U	32	0-4	U	U	0-4	U	U	U	U
V	33	0-5	V	V	0-5	V	V	V	V
W	34	0-6	W	W	0-6	W	W	W	W
X	35	0-7	X	X	0-7	X	X	X	X
Y	36	0-8	Y	Y	0-8	Y	Y	Y	Y
Z	37	0-9	Z	Z	0-9	Z	Z	Z	Z
)	40	11-5-8	)	None(3)	12-4-8	[	)	)	)
-	41	11	-(5)	-(5)	11	-(5)	-(5)	-(5)	-(5)
+	42	12-0	+	None(3)	12	&	+	+	+
<	43	12-6-8	<	None(3)	12-6-8	<	None(3)	<	<
=	44	0-5-8	=	None(3)	3-8	#	=	=	=
>	45	6-8	>	None(3)	6-8	>	None(3)	>	>
&	46	12	&	+	2-8	None(3)	None(3)	&	&
$	47	11-3-8	$	$	11-3-8	$	$	$	$
*	50	11-4-8	*	*	11-4-8	*	*	*	*
(	51	12-5-8	(	None(3)	0-4-8	%	(	(	(
%	52	0-4-8	%	(	0-5-8	=	None(3)	%	%
:	53	5-8	:	None(3)	5-8	:	None(3)	:	:
?	54	12-7-8	←	None(3)	12-0	+	None(3)	Note 6	?
!(Exclaim)	55	11-0	X(Times)	None(3)	11-0	X(Times)	None(3)	!	!
,(Comma)	56	0-3-8	,(Comma)	,	0-3-8	,	,	,	,
\	57	2-8	None(3)	None(3)	0-6-8	]	None(3)	\ (shift L)	\
0	60	0	0	0	0	0	0	0	0
1	61	1	1	1	1	1	1	1	1
2	62	2	2	2	2	2	2	2	2
3	63	3	3	3	3	3	3	3	3
4	64	4	4	4	4	4	4	4	4
5	65	5	5	5	5	5	5	5	5
6	66	6	6	6	6	6	6	6	6
7	67	7	7	7	7	7	7	7	7
8	70	8	8	8	8	8	8	8	8
9	71	9	9	9	9	9	9	9	9
'(Quote)	72	0-7-8	11	None(3)	4-8	@	-(4)	'(Quote)	'(Quote)
;	73	11-6-8	;	None(3)	11-6-8	;	None(3)	;	;
/	74	0-1	/	/	0-1	/	/	/	/
.(Period)	75	12-3-8	.	.	12-3-8	.	.	.	.
□	76	4-8	@	-(4)	0-7-8	"	None(3)	"	"
≠ or stop (2)	77	0-2-8	≠	None(3)	0-2-8		None(3)	Cntrl C	None(2)

NOTES:

1) The @ is the control card flag if it appears in column 1 of any card.

2) The ≠ will not be printed on the high speed printer or the teletype. On these two devices the ≠ character acts as the line termination character.

3) None means that there is no such key. This hole pattern must be multipunched.

4) This is the underscore character that is under the = sign on the 026 keypunch.

5) This is the minus character and is marked SKIP on the 026 keypunch. Either upper or lower shift may be used.

6) Normally, the ? key causes deletion of the current line. To enter ? as an input character, press ESC (ESCAPE), then the ? key.

APPENDIX B

DIAGNOSTIC MESSAGES

I. CONTROL STREAM DIAGNOSTIC MESSAGES

The following messages are among the most common and typical of the many hundreds in the system. A large number of other messages are worded somewhat differently, but have meanings which are similar to these.

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. Note that most of the messages issued by the FURPUR processor correspond to a specific I/O error and status code.

When a twelve-octal-digit status code is given in an error message, it often has bit settings corresponding to one or more of the causes of facilities rejection (FAC REJECT) or facilities warning (FAC WARNING) described under @ASG in section 4.5.2.6.

Some diagnostic messages refer to operator response keyins. Here are the usual meanings of the most common operator response keyins:

A	Try again with standard recovery.
B	Return I/O status 12 to packet.
D	Declare device down.
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.
H	Halt the operation.
I	Initiate a locked out or suspended symbiont.
L	Lock out a symbiont.
N	The reply is "no."
Q	Re-enter a symbiont file in its appropriate queue.
R	Reprint or repunch a symbiont.
S	Suspend a symbiont.
T	Terminate a symbiont.
X	Abort a symbiont, or abort a run.
Y	The reply is "yes."

One of three abbreviations, 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 call statement, such as @ASM, @COB, @FOR, or @MAP. For processors such as @ELT which have no RO subfield, only SI and SO are meaningful.

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.
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.
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.
@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.
nn ILLEGAL CHARACTER C	The coarse scheduler encountered the illegal or badly positioned character C at column nn of the above control statement.
INTERVENING STATEMENTS SKIPPED	A conditional statement has been encountered and has caused one or more control statements to be bypassed.
RUN KILLED VIA AN X-KEYIN	The operator replied with an X to @MSG control statement with a W option; batch runs are terminated in this case. The operator typed an unsolicited X keyin for this run. An unsolicited X keyin for a demand run simply terminates the currently executing program.
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: 1) Excessive number of granule tables (change track granularity to position granularity). 2) Excessive number of activities (check for ER FORK$ loop). 3) Excessive number of files assigned (check for ER CSF$ loop).

PMD NOT ALLOWED	@PMD 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 @PMD's of any programs are allowed.
PROGRAM TOO LARGE	There is not enough space available in the user area of main storage to load a program that is this large or the D bank cannot be loaded because B_S + D bank size is greater than the highest available address. The hardware does not support negative B_I or B_D.
RUNSTREAM ANALYSIS TERMINATED	The run has been terminated because of an error condition and the remaining control statements are not processed.
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.
TIME ESTIMATE EXCEEDED	The total central processor time used by the run exceeds the estimate in the RUN-TIME field of the @RUN statement. Batch runs are terminated; demand runs may continue by entering another control statement.
USER DID AN ER EABT$	The running program requested an error abort. All activities are terminated. Batch runs are terminated following any post-mortem dumps. This message is usually preceded by another message giving the reason for the error abort.

II. ERROR CODE MNEMONICS

Contingency	Contingency Type	Mnemonic
Illegal Operation	1	IOPR
Guard Mode (see section 1.5)	2	IGDM
Floating Point Overflow	3	IFOF
Floating Point Underflow	4	IFUF
Divide Overflow	5	IDOF
Restart	6	IRST
Abort	7	IABT
Console Interrupt	10	IINT
Test and Set Interrupt (R/T only)	11	ITS
'ERR MODE' Entry	12	IERR$

Error Name	Error Type	Mnemonic
I/O Call Error	1	I/O
Symbiont Call Error	2	SYMB
ERR$ Call	3	ERR$
Illegal or Bad ER	4	ER
Console Call Error	5	CONS
Communications Errors	6	COM2
Communications Errors	7	COMM
Reentrant Processor Call Error	10	REP

III. 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 following list is the full set of defined ERR mode codes, with two exceptions:

1) 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.

2) Types 6 and 7 (communications) codes are not included because they are lengthy and not used by most programmers.

Some of the code definitions refer to obscure, not-yet-implemented, or to-be-dropped ER's which the user will probably not encounter.

Type	Octal Code	Description
1	0	Normal I/O completion without complications.
1	1	End-of-file (EOF) encountered on read or search function. The word count actually transferred is supplied in H2 of the fourth word of the I/O packet.
1	2	End-of-tape mark encountered on magnetic tape on a read backward from load point or on a write.
1	3	No find was made on a mass storage device search.
1	4	A nonintegral block was read from magnetic tape. The number of data characters accepted from the last word is indicated in the AFC (abnormal frame count) field in S3 of the fourth word of the I/O packet.
1	5	An attempt was made to search or read from an unassigned area of mass storage. If the starting address is legal, the read is truncated as reflected by the word count in the I/O packet.
1	7	Timeout on an absolute I/O drum read (ABSR$) or write (ABSW$).
1	10	Fastrand mass storage file timed out before being unlocked.

Type	Octal Code	Description
1	11	A nonrecoverable error has occurred and either the suppress recovery mode is set for magnetic tape or an answer of G was given to an error message.
1	12	A read or write error on magnetic tape has resulted in loss of position on the unit.
1	13	I/O attempted on peripheral unit declared down.
1	17	Reference made to an unassigned file (EXEC 8 only).
1	20	Write or area release attempted for file in read-only mode, or read attempted for file in write-only mode.
1	21	Reference made to an unassigned file.
1	22	Attempt to write beyond assigned area of a mass storage file. This is a very common error, and results when the maximum granules subfield on the @ASG control statement is not set large enough. When this subfield is not specified, a system standard such as 2 positions = 128 tracks = 8192 sectors is used. Note that adding elements to a program file, such as through a @COPIN or @ASM control statement, is equivalent to doing a write operation.
1	23	The I/O packet whose address is given in register A0 is wholly or partially outside of the program's I or D bank limits.
1	24	The requested I/O function is not defined for the assigned equipment type, or there is a noncompatible field on a set mode request.
1	25	The buffer which is defined in the user's I/O packet is wholly or partially outside of the program's I or D bank limits. For GW$, SCR$, and SCRB$ functions, this error code is given if the number of access words is zero or more than 50, or if the total word count is more than 65K.
1	26	Illegal interrupt routine starting address.
1	27	An I/O request was made with the status field of the I/O packet set negative, indicating that a previous I/O request which references this packet has not yet been completed. The distinction between ER IO$ and ER IOW$ is that the latter requests a wait until I/O completion before returning control to the user program. If there is doubt about a previous I/O function having been completed, an ER WAIT$ ensures delay until completion.

Type	Octal Code	Description
1	30	The interrupt activity specified is greater than 35 or is already in use.
1	31	A magnetic tape operation which specified user recovery did not furnish an interrupt activity.
1	32	User program changed I/O packet prior to completion of an I/O request.
1	33	Fastrand-format I/O request not initiated because it would cause this run's program control table (PCT) to expand past its maximum. An entry must be kept for each granule that has been written into for every Fastrand-format file that is assigned to the run.
1	34	An absolute write, an absolute read with illegal or alternate subsystem number, or an absolute read on a non-mass-storage subsystem was attempted.
1	35	A second read and lock (RDL$) request by an activity for a particular area, or an unlock (UNL$) request for an area that the activity had not previously locked.
1	36	An ER WAIT$ was not immediately preceded by a Test Positive instruction without h and i designators, or a WAIT$ or WANY$ request was made without a previous outstanding I/O request.
1	40	Previous I/O request is still in process.
2	1	READ$ attempted past the end of file.
2	2	Additional READ$ attempted after receiving an A0 bit 35 status indicating that next control statement in run stream has been encountered.
2	3	I/O error encountered by READ$.
2	4	Attempt to READ$ a standard data format (SDF) file or element with Fieldata image length exceeding 15 words, or ASCII image length exceeding 20 words (EXEC 8 control only).
2	5	@ADD control statement in control stream cannot be processed due to an error. An additional diagnostic message is given.
2	6	The READ$ packet whose address is given in register A0 is wholly or partially outside of the program's I or D bank limits.
2	10	Attempt to @ADD by means of CSF$ of an element from tape.
2	11	Nested levels exceed maximum allowed on @ADD from CSF$.

Type	Octal Code	Description
2	12	The file referenced on an @ADD by means of CSF$ is not assigned or catalogued.
2	13	Element referenced on an @ADD by means of CSF$ not found in referenced file.
2	14	Nested loop on @ADD by means of CSF$.
2	15	Attempted @ADD by means of CSF$ from equipment other than Fastrand mass storage.
2	16	@ADD by means of CSF$ cannot assign catalogued file because it would cause this run's PCT to expand past its maximum.
2	20	Attempt to reference an unassigned alternate symbiont file by a demand or real-time run.
2	21	Alternate file could not be assigned for PRINT$ or PUNCH$.
2	22	Alternate symbiont ER request is improper for type of file.
2	23	The alternate symbiont file packet whose address is given in register A0 is wholly or partially outside of the program's I or D bank limits.
2	24	The read alternate (READA$) file being referenced has not been assigned.
2	25	Bad format encountered in first call to a READA$ file.
2	26	Alternate file is not Fastrand-formatted mass storage or tape file.
2	27	Maximum number of alternate files, as specified by systems generation parameter SMALTM, has been exceeded.
2	30	Maximum number of breakpoints for PRINT$ or PUNCH$ exceeded.
2	34	Length of print alternate (PRNTA$) file has been exceeded.
2	35	Length of punch alternate (PNCHA$) file has been exceeded.
2	36	Length of PUNCH$ file has been exceeded.
2	37	Length of PRINT$ file has been exceeded.
2	40	The buffer which is referenced in user's print or punch packet is wholly or partially outside of the program's I or D bank limits.
2	41	The maximum pages specified in @RUN control statement (or system standard, if none was specified) has been exceeded.

Type	Octal Code	Description
2	42	The maximum cards specified in @RUN control statement (or system standard, if none was specified) has been exceeded.
3	-	Since error type 3 results simply from a direct call on ER ERR$, there are no type 3 error codes.
4	1	An ER was attempted with an ER function code beyond the range of currently defined ER's, or with a code reserved for use by EXEC 8. A list of currently defined ER's is maintained in system relocatable library. SYS$*RLIB$, element ERU$, CSF$, for example, is equated to the ER code 17_8. These equates are listed in most L option @MAP listings under EXTERNAL DEFINITIONS.
4	2	The ER packet whose address is given in register A0 is wholly or partially outside of the program's I or D bank limits.
4	3	Illegal ER function code within range of those defined.
4	4	Improper identity supplied upon AWAIT$ request.
4	5	Activity number (ID) supplied on FORK$ request is either out of range or in use.
4	6	This run's account number does not permit requested real time priority on RT$ or FORK$ call.
4	7	Code 7 formerly indicated that the maximum time specified in @RUN control statement has been exceeded. This condition now causes a direct run abort, without reverting to error mode.
4	10	The FACIL$ packet whose address is given in register A0 is wholly or partially outside of the program's I or D bank limits.
4	11	I/O mass storage read error, or bad information in run's PCT, while processing FACIL$.
4	20	Bad packet or file control table (FCT) address on BBEOF$.
4	21	The referenced file is not mass storage format, or it is not catalogued.
4	30	Requested NRT$ from an activity which is not real time.
4	31	Illegal creation of a real time activity by FORK$.
4	32	The activity marked as named on a NAME$, ACT$, or DACT$ request has not been properly defined.

Type	Octal Code	Description
4	33	Illegal II$ request.
4	37	The tape file referenced by TSWAP$ or TINTL$ is not assigned.
4	40	Syntax error discovered in control statement image furnished to CSF$.
4	41	Image length of over 40 words specified for CSF$.
4	42	Control statement image contains a command which cannot be serviced by CSF$.
4	43	The control statement image whose address is given in register AO is wholly or partially outside of the program's I or D bank limits, on CSF$ request.
4	44	The number of @LOG control statement entries submitted exceeds maximum allowed by CSF$.
4	46	LOAD$ of relocatable segment (RSEG) cannot be executed because it would cause the run's PCT to expand beyond its maximum.
4	50	Nonzero I/O status on ER LOAD$ of segment.
4	51	Attempt made to LOAD$ an undefined segment.
4	52	The segment load table (SLT$) at start of user's D bank contains bad information, and has possibly been overwritten. LOAD$ has encountered an illegal segment definition, or segment limits outside I or D bank.
4	53	Invalid MCORE$ request.
4	54	LCORE$ request to release main storage not currently held.
4	55	Request by means of MCORE$ for more main storage than is available.
4	57	Attempt to release a contingency or a re-entry address by means of LCORE$.
4	60	Bad recognition key on DLOC$ or DIW$.
4	61	Bad packet for SNAP$.
5	0	The COM$ packet whose address is given in register AO is wholly or partially outside of the program's I or D bank limits.
5	1	The output buffer which is defined in the user's COM$ packet is wholly or partially outside of the program's I or D bank limits.

Type	Octal Code	Description
5	2	The expected input count field for a type and read operation exceeds 50 characters.
5	3	The input buffer which is defined in the user's COM$ packet is wholly or partially outside of the program's I or D bank limits.
10	1	The RLIST$ packet whose address is given in register AO is wholly or partially outside of the program's I or D bank limits.
10	2	REP's entry point was defined as zero (due possibly to being undefined at @MAP time), and an attempt was made to link to it by means of LINK$ or RLINK$.
10	3	File is not assigned or not on mass storage on an RLIST$ request.
10	4	RLIST$ entry name not found.
10	5	REP's I bank length exceeds the program's D bank starting address on a LINK$, RLINK$, or RLIST$ request.
10	6	Attempt through LINK$, RLINK$, or EXLINK$ to attach multiple REP's to the same program.
10	11	RLIST$ request to remove previous REP list while other REP's are still active.
10	12	Specified REP name not found in system search on a LINK$ or RLINK$ request.
10	14	EXLINK$ or UNLNK$ request is not from a linked routine.
10	15	The number of RLIST$ REP names exceeds the system's maximum.
10	16	LINK$, RLINK$, or RLIST$ request not initialized because it would cause the run's program control table to expand beyond its maximum.
10	17	The system detected an I/O error in loading a REP, or on a LINK$, RLINK$, or EXLNK$ request.
10	20	The main program plus the REP's core requirements exceed total user main storage availability.
10	21	Attempt to change REP size by MCORE$ or LCORE$.
10	22	Same as code 20 except occurred because of main storage fragmentation due to downed main storage or real time programs. The effect of an ER UNLNK$ will have occurred before control is returned to a program's contingency routine.

APPENDIX C

STANDARD TAPE TRANSLATION (BCD-FIELDDATA)

Tape to Processor

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

Processor to Tape

CPU Code	Tape Code	CPU Code	Tape Code
00	17	40	74
01	75	41	40
02	55	42	60
03	77	43	76
04	57	44	13
05	20	45	16
06	61	46	00
07	62	47	53
10	63	50	54
11	64	51	34
12	65	52	35
13	66	53	15
14	67	54	72
15	70	55	52
16	71	56	33
17	41	57	36
20	42	60	12
21	43	61	01
22	44	62	02
23	45	63	03
24	46	64	04
25	47	65	05
26	50	66	06
27	51	67	07
30	22	70	10
31	23	71	11
32	24	72	14
33	25	73	56
34	26	74	21
35	27	75	73
36	30	76	37
37	31	77	32