SPERRY UNIVAC
1100 Series
Operating System
(EXEC Level 35R1)

Installation Reference

SPERRY UNIVAC
COMPUTER SYSTEMS

UP-8486.1

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

Sperry Univac is a division of Sperry Rand Corporation.

PRINTED IN U.S.A.

Page Status Summary

Issue: UP-8486.1

Section	Pages	Update
Cover/Disclaimer		
PSS	1	
Contents	1 thru 11	
1	1,2	
2	1 thru 13	
3	1 thru 23	
4	1 thru 4	
5	1 thru 41	
6	1 thru 4	
7	1 thru 58	
8	1 thru 92	
9	1 thru 63	
User Comment Sheet		
Total: 314 pages and cover		

Section	Pages	Update

Section	Pages	Update

Contents

Contents

Page Status Summary

FIGURES

TABLES

105

1. Introduction

1.1. GENERAL

SPERRY UNIVAC 1100 Series systems come in a wide range of sizes, configurations, and powers, all using the same basic SPERRY UNIVAC 1100 Operating System (Operating System, OS). A great deal of flexibility in the use of the 1100 Series standardized components is offered. This manual has been prepared to help the installation manager make the most effective use of the system by describing the aids and procedures Sperry Univac provides for this purpose.

1.2. SCOPE

This manual will show options available in configuring software, and in solving software problems by the site personnel, and aid the installation manager in directing the operating staff.

1.2.1. Installation Manager Defined

The term "installation manager" as used in this manual is the person or persons responsible for the overall operation of the site. This will include directing the operating staff, hardware/software configuring, scheduling production, security, and integrity of the system.

This manual contains some information that concerns only the installation manager,

NOTE:

It is the responsibility of the manager to control access to this manual.

1.2.2. Level of Information

A basic knowledge of the Operating System software and hardware is assumed. System analysis experience, though desirable, is not mandatory. This manual is not intended to be used as an instructional textbook. Wherever possible, references will be made to other Sperry Univac publications for more detailed information.

Whenever possible the subjects covered will be at the latest released Executive level. Sites running at levels previous to the latest released level may experience problems using the tools and/or aids presented.

It is not the intention of this manual to establish Sperry Univac policy. Its only purpose is to assist the manager in the operation of the installation.

1.2.3. Hardware

Some Customer Engineering Department functions and responsibilities will be discussed, along with offline/online test programs of the Peripheral Test Sequencer (PTS), software system, and TRACE (Total Remote Assistance Center).

1.2.4. Software

The Operating System was designed and is maintained with the following basic objectives in mind:

- Manage the resources of the system so as to use them most efficiently. (For highest productivity).
- Provide a competitive and evolving set of functional capabilities.
- Provide a simple, straightforward interface for the user.
- Provide an environment to handle the complex requirements of sophisticated users.
- Support batch, remote batch, demand, real time, and time sharing users concurrently and with equal efficiency.
- Readily evolve to encompass new hardware and software components as they become available.
- Optimize facility usage with extensive variations in configurations.
- Provide an extensive user utility package.
- Allow system management by the owner through a priority structure.
- Allow control of all system resources with respect to access, usage, accounting and billing.
- Continually upgrade the reliability of the system throughout the lifetime of the product.
- Create a system that can be maintained in a reasonable manner.

Software is the most volatile part of the overall system, hence the major portion of this manual will discuss software components.

The sections on System Security (Section 8) and Logging and Accounting (Section 9) should give the installation manager an overview of the methods of control which can be exercised over the production and security of the system.

2. Product Support Services

2.1. MARKETING SUPPORT

Sperry Univac's local Marketing and Services organizations provide software support to their customers. Sperry Univac sale, lease, and rental contracts for computer systems are "bundled". That is, standard software is provided and maintained as part of the basic contract. Local Sperry Univac software specialists assist customers in maintaining Sperry Univac supplied software and in implementing new or enhanced levels of this software as it is released. The SPERRY UNIVAC 1100 Series Operating System, (Operating System), including all its language compilers and other processors is, of course, fully supported, as are many other software systems and capabilities.

For the installation and implementation of some computer systems, Sperry Univac appoints a project manager to manage its responsibilities. During the time there is a project manager, all requests for software support should be made through the project manager. For all other situations, requests should be made through the Sperry Univac account representative.

In most cases, Sperry Univac's local organization will have the capabilities and resources for providing the software support needed. However, support groups and pools exist in several echelons in Sperry Univac's field marketing structures and in the development centers specifically for the purpose of providing specialized knowledge needed for detailed support of all supported software. In addition, these support groups are in a position to fix system bugs whose symptoms may have been detected anywhere in the world.

Local Sperry Univac branch offices provide most of the software support to customer systems with the support groups providing the remainder. In most cases the support is routine, initiated and conducted by local Sperry Univac personnel. In those cases where the customer initiates the request for support, the customer should always contact his Sperry Univac project manager or account representative (salesman). In those situations in which the services of a support group are needed, the project manager or account representative will use established Sperry Univac procedures and channels to obtain it. Keep this in mind when reading further in this section. Procedures for the Software User Report, Technical Question, Request for Change, and educational material ordering are to be followed by Sperry Univac's field personnel in response to customer needs, not by the customer directly. In this way, Sperry Univac can assure that all requests for support are met in a timely manner and by the best qualified personnel.

2.2. SOFTWARE SUPPORT

2.2.1. The Standard Library

When a user document is produced for one or more of the 1100 Series Systems, several decisions are made as to its categorization. The first such decision is whether the manual should be classified as a Standard Library Item (SLI) or a Restricted Distribution (RD) item. Manuals in the latter class are only available on a specific order. Standard Library Item manuals are provided automatically, including revisions and updates. The decision as to whether to classify a manual as Standard Library Item is made on the basis of a number of factors, the most important of which are:

1. an assessment of how widespread, or general, its use will be in the 1100 Series user community, and

2. input from marketing organizations as to demand and potential for the manual.

When a new customer is "initialized," the branch responsible for servicing that site will submit a memo to the Sperry Univac Customer Information Distribution Center, listing the names of personnel the customer wants on the automatic mailing lists.

Upon notification, the local Sperry Univac marketing organization will submit a Customer Information Distribution Center Requisition (UDI-578) specifying that a set of 1100 Series manuals are to be shipped to the customer.

Whenever a Standard Library Item manual is revised, the manual is reprinted in its entirety and a complete replacement copy, along with a descriptive library memo, is automatically shipped to each name on the mailing list. When such a manual is updated, an update package is prepared containing only changed or additional pages, and this package, with a library memo, is shipped to each name on the mailing list.

For Restricted Distribution manuals, the descriptive library memo is handled as a Standard Library Item. This means that a one-page description of a new Restricted Distribution manual, or a revision or update to a Restricted Distribution manual, is automatically shipped to holders of the Standard Library. On receiving this library memo, the described material may be ordered through the Sperry Univac branch office, using the Customer Information Distribution Center Requisition (UDI-578).

It should be noted that whether a manual is classified as a Standard Library Item or Restricted Distribution, it has been copyrighted.

The procedure discussed here applies to USA customers. For international customers, the same general principles apply, but the routing is slightly different. Locations outside the United States should route requests through their subsidiaries, which direct their correspondence to the Publications Dissemination Department in Blue Bell, Pennsylvania. A special ordering form, UD1-1371, is available for this purpose.

2.2.2. SPERRY UNIVAC 1100 Series Operating Systems Documentation

Sperry Univac Systems Software Development uses three primary vehicles for disseminating Operating Systems information to the user. These are:

- Sperry Univac Systems Publication (UP) Documents
- Software Release Documents (SRD)
- Field Support Bulletins (FSB)

The purpose of each class of documentation, how they relate to each other, and how to effectively use the documents are the subjects of this section.

The UP documents are the formal printed manuals which are widely distributed through Sperry Univac's Customer Information Distribution Center. These manuals, which include both hardware and software documentation, provide the base for all user documentation. Examples of manuals in this class are:

SPERRY UNIVAC 1100 Series Executive, Volume 2, EXEC Programmer Reference UP-4144.2, (current version)

SPERRY UNIVAC 1100 Series, FORTRAN (ASCII), Summary UP-8245 (current version)

The document, "SPERRY UNIVAC 1100 Series Summary of Current Documentation," (UP-7893), which is updated periodically, provides a categorized listing of current UP documents.

The UP manuals must conform to more formal documentation requirements, and they have a much wider distribution than SRDs and FSBs. Therefore, the time required to produce and distribute the manuals is much greater for the UP as opposed to the other documents.

UP documents vary as to the coverage of the subject material:

Type 1: Introductory Manual

Scope: Contains an introductory overview of a particular concept for those wishing to understand its purpose and general functional capabilities without necessarily wishing to know how it works or how to use it.

Type 2: Subject Manual

Scope: Contains information, procedures, illustrative material, and relevant examples for those wishing to learn how to apply the concept. Essentially, it explains what the system is and how you use it. It does not get into details of internal construction.

Style: The style is conversational and has a tutorial orientation.

Level: Assumes a knowledge of data processing concepts, techniques, and terminology but no knowledge of the subject software or hardware element.

Type 3: Reference Manual

Scope: Contains information in ready reference form for those wishing quick accessibility to information.

Style: The style is concise and terse and generally void of illustrative material.

Level: Assumes a detailed knowledge of data processing concepts, techniques, and terminology as well as total familiarity with the subject system.

The Field Support Bulletins (FSB) are internal memoranda from Systems Software Development to Sperry Univac Field Marketing project personnel. The information in the bulletins is not expressly intended for distribution to the customer. However, the Sperry Univac personnel distribute appropriate information to the customer as necessary.

The FSB is used to disseminate user programmer information between major system releases and to provide answers to Technical Questions (TQ). Detailed information would pertain to processor updates, incremental system releases, etc., and not to the release of a major system. The appropriate FSB information is incorporated into updates of the UP documents.

The appropriate FSBs provide timely documentation of the most recently released software. It is the responsibility of the installation management to determine which portions of the documentation are needed by the various programming groups.

2.2.3. Software Release Notification and Support Documentation

Three types of documentation are used to support software releases. They are:

- Software Release Announcement (SRA)
- Software Release Documentation (SRD)
- Technical Documentation (TD)

The first of these, the SRA, is used to inform Field Marketing that a new or enhanced version of a software product is ready for delivery. The software package is, when desired, ordered using normal procedures.

When the order for the software is filled, the software package will include necessary reference information (SRD) for the generation, installation and operation of the release involved.

Technical Documentation can be requested from the Software Support group, when required. The 1100 Series Executive Technical Documentation is available on microfiche only.

NOTE:

Software for Executive levels 35R1 and higher is proprietory.

2.2.4. Communications with Systems Software Development

Software Continuation and Support provides the interface between Field Marketing and Systems Software Development. All technical communications should be directed to your Sperry Univac Project Manager or Account Representative who will work through this group. These include the following items:

1. Critical customer problems with standard Sperry Univac software.
2. General inquiries submitted via the Technical Question (TQ) form.

3. Software User Reports (SUR).

4. Requests for Change (RFC).

5. Field Software Orders (FSO)

2.2.4.1. Technical Questions (T.Q.)

Questions relating to the technical aspects of the 1100 Series Operating System are submitted on Sperry Univac Technical Question forms by your Project Manager or Account Representative. This form should *not* be used to inquire about the following:

1. Scheduled software implementation dates. These should be written inquiries to the appropriate Project Manager or Account Representative submitted through the Marketing branch office.

2. Suspected software malfunctions.

3. Reiteration of documentation available in other publications.

4. Characteristics of hardware operation.

5. Debug of user generated code.

6. Requests for implementation of special software enhancements. These should be submitted via RFC (Request for Change) forms.

Answered TQs are published and distributed periodically as Field Support Bulletins (FSB).

2.2.4.2. Request for Change (RFC)

The RFC is the formal means of requesting enhancements or changes to the Operating System (including processors). The RFC must be prepared by Sperry Univac Field Marketing personnel.

Suggested modifications should be of a general interest to all users and not for a specific application. Most changes of this type require long-term planning for implementation.

Because this is a formal request, the Marketing branch office will be informed as to its disposition.

2.2.4.3. Software User Report (SUR)

The Software User Report (SUR) System was developed to assist users and Field Marketing representatives in reporting software malfunctions. Although Sperry Univac attempts to insure that the software is error-free before distribution to customers, malfunctions may occur for many reasons. When such a problem arises, it should be reported in a timely manner, so that a correction can be made and distributed to the entire customer base. Even though a temporary solution may be found locally, the problem should be brought to the attention of the appropriate Software Development Center. In reporting, the user should work in conjunction with the local Sperry Univac Marketing representative.

It is the user's responsibility to clearly define a system malfunction since today's software systems are extremely complex. Therefore, adequate documentation and information concerning the malfunction or problem must be supplied. Every attempt should be made to localize the problem and to describe the symptoms adequately. When reporting Operating System failures, a complete

description of the hardware and software operating environment (multiprogramming, devices active, programs active, operator actions, etc.) should be provided, along with a comprehensive review of the problem symptoms. Programs which have been determined to be the cause of a system malfunction should be reduced to the minimum size which will still execute and demonstrate the problem. For errors occurring during compilation or assembly, the source listings and a copy of the source program or a portion thereof should be provided.

SURs are submitted when program malfunctions occur and it is suspected that the trouble lies in the standard software. When submitting a SUR, please use form UD1-745. Instructions for completing and submitting SURs are printed on the back of this form, in UP-8019, and in Field Support Bulletin 259.

When submitting a SUR to Software Continuation and Support through Field Marketing channels, the following guidelines should be observed:

1. The Systems Analyst should review each SUR to determine that:

   a. User code is not the cause.
   b. A list of user code is sent with the panic dump.
   c. Hardware is not at fault.
   d. It is a legitimate SUR, not an RFC or TQ.
   e. A complete panic dump is included.
   f. The SUR is filled out completely.
   g. List of Change Control Forms (CCFs) installed beyond the released level is included.

2. On the SUR form, be as specific as possible in describing the problem. Include as many surrounding circumstances as possible (i.e., console locked, jobs not terminating, ... etc.).

3. Isolate problems and include card decks which cause the problem to occur.

4. Include a console sheet or copy thereof reflecting the system's status and activities at the time of the error.

2.2.4.4. SUR Status Reports

The SUR Status Report is distributed periodically to all Marketing branches and subsidiaries supporting 1100 Series sites. The purpose of this report is to provide the status of all unanswered SURs submitted and to serve as Roseville Software Development Center acknowledgement of the receipt of a SUR and its entrance into the problem analysis process.

Appropriate portions of this report will be distributed to designated site personnel.

Listed in the report is the Roseville Software Development Center (RSDC) assigned number, date of receipt at RSDC, and customer reference number for all unanswered SURs. The problem description is included for those SURs received during the report period. Subsequent reports will also list the RSDC assigned register number of all SURs answered and returned since the previous report. The information in this report is sorted by submitting site, software component, release level, and RSDC assigned register number. Also included is an estimated completion date (ECD) for each SUR based on the priority and software component. An expiring ECD will be assigned a new date and include a reason for expiration.

The SUR Summary Report is issued biweekly and indicates all activity, including both open and closed SURs, that have been submitted to the Development Centers. Closed SURs include definitive corrective information (CCFs), that answers the SUR. This report is currently available in microfiche form. An index frame on the lower right-hand corner of the microfiche references each frame on the

fiche. The CCF portion is divided into "old" and "new" CCFs; the new CCFs representing only those not identified in previous SUR Summaries.

2.2.4.5. Software Ordering

Software Release Announcements (SRA) are sent to the local Sperry Univac representatives with the 1100 Series Field Software Order form attached. When ordering software, this form should be used. The form must be signed by the local Sperry Univac representative. Form UD1-1851 may be used instead of the 1100 Series Field Software Order form (see UP-8297).

All SPERRY UNIVAC 1100 Series System supported software should be ordered through your local Sperry Univac representative from Software Order Services in Roseville. New installations should contact World Wide Software Support for software at least three weeks prior to installation.

2.2.5. Educational Materials

Sperry Univac personnel associated with your site also have access to numerous educational materials.

The Sperry Univac Education Center has compiled a document to provide a reference catalog of all education materials available. Designed to increase awareness of Sperry Univac's many educational materials, this publication offers brief descriptions of the various tools which can be effectively incorporated into a class, seminar or workshop, depending upon the user's requirements.

Sperry Univac also offers video seminars with the view in mind of providing more timely information. Video production is now being coordinated with manual and product releases. Release seminars are being developed to provide more timely release of information on new software releases. The seminars employ a discussion format as the major communication vehicle.

There are also instructional seminars for Sperry Univac products.

2.3. HARDWARE SUPPORT

2.3.1. Error Editor (EDTERR)

The EDTERR (Error Editor) program provides the ability to edit the log file from the Operating System. From the edited data, a report is presented in printed format covering a history of the type 06 and type 27 Errors logged by the Operating System plus a report on the system and associated devices using data from the Master Configuration Table (MCT) and Mass Storage Table.

The printed report contains Device Name, Control Unit Name, Device Address, Channel Address, Input/Output Unit Number, Channel Command Words, Channel Status Words, Device Status, Date, Time, Number of References Since Last Error, Run-id, and type of device responsibile for the error.

For more detailed information see SPERRY UNIVAC 1100 Series Error Log Editor (EDTERR) DA3019. This is a Customer Engineering publication.

2.3.2. PTS Software System

2.3.2.1. Introduction

This general description acquaints the reader with the essential features of the Peripheral Test Sequencer (PTS) software system for the SPERRY UNIVAC 1100 Series peripherals. The following paragraphs contain descriptive information intended to define how certain portions of the system operate, and also to describe the overall interactions of the various components of the system, and general philosophy which is intended to familiarize the reader with the general approach towards peripheral maintenance and testing which underlies the development of this system.

- Peripheral And System Maintenance Procedures

The maintenance of SPERRY UNIVAC 1100 Series Systems can be divided into five categories: maintenance surveillance, concurrent corrective maintenance, offline corrective maintenance, preventive maintenance, and emergency maintenance.

- Maintenance Surveillance

Maintenance Surveillance applies to all mass storage, tape units, card equipment and other equipment which generate recoverable error interrupts.

All recoverable errors are recorded in the Executive system log, along with the time of occurrence, number of operations since the last recoverable error, device which failed, and channel number. This information is available to the Customer Engineer through a log edit routine.

Peripheral devices are subject to performance deterioration primarily due to mechanical deterioration. Examples are dirt on a tape unit head, a bent card, a damaged surface on a drum unit, etc. The Executive recoverable error log permits maintenance surveillance of this equipment without running a test program. With the recoverable error log edit a Customer Engineer can quite effectively sort out the equipment problems from the operational problems such as bad tape, etc.

Malfunctioning equipment discovered by maintenance surveillance should then undergo corrective maintenance.

- Concurrent Corrective Maintenance

This term applies when the Customer Engineer requires support from a test program for a malfunctioning device. For this purpose, a set of online maintenance aids and peripheral tests is included in PTS. To be more accurate, these online maintenance aids and tests form the basis of PTS.

- Offline Corrective Maintenance

In some cases, malfunctioning devices may be repaired without support from the rest of the system by hardware test sequences. An example is the Type 7013 unitized storage which has a built in theoretical worst-case generator and worst-case cycler. The tape, card, and printer equipment also have offline hardware test sequences, which facilitate repair of some malfunctions without involving other parts of the system.

- Preventive Maintenance

This term defines a scheduled time where a part of the system is dedicated to Customer Engineering. During this time, the Customer Engineer will use the offline PTS boot tape which will automatically adjust to the central complex into which it is being loaded. That is, if the maintenance activity has

one storage unit, one central processor and one IOAU or IOU, PTS will adjust to this configuration. If the bootstrap load occurs on channel 5, the boot loader will adjust to this.

After the boot load is complete all Stimuli Response Lists (SRL) from the boot tape are in storage. No further loading is necessary since PTS is a single program with the capability of testing all 1100 Series peripherals accessible from the IOAU or IOU.

Although it is desirable to eliminate the type of preventive maintenance which requires dedicating all or part of the central complex to maintenance, this type of capability must be provided to accommodate the emergency maintenance situation. Maintenance surveillance activities can take over much of the preventive maintenance activities of the past.

- Emergency Maintenance

This term applies to unscheduled maintenance which requires that a part of the central complex be downed in a way which may severely impact the user application.

2.3.2.2. The Effect of PTS on Maintenance Procedures

As efforts become more successful in eliminating the need of dedicating a part of the central complex to preventive and emergency maintenance, the need for continuity of design, from the dedicated maintenance test software to the online test software, becomes more acute.

In constructing an offline PTS boot tape, a group of program elements that replace the Executive are included in the collection. These elements are the only difference between offline and online PTS.

By limiting the Executive functions used, it is possible to limit the size of the PTS Executive level code, which sustains the multiple activity multiprocessor environment to under 4K instructions. Additional Executive level instructions are required for traps to allow manifesting system failures as soon after they occur as is possible.

The maintenance activities which impact the user applications least are offline and online corrective maintenance and maintenance surveillance. The goal with PTS is to move as many maintenance activities as possible into these three categories.

2.3.2.3. PTS Software System Characteristics

The software system is divided into the following two parts:

1. Tabled information that defines the testing sequence.

2. Coding that performs actual testing and handles all utility functions required.

The tabled information defining the test sequence is realized by assembly of English language statements that define function and data action for testing given subsystems. These statements combine to form test sequences which are called Stimuli (for functions and data sent) and Response (for interrupts and data returned) Lists.

The Stimuli Response Lists (SRL) in source form (English) are almost completely self documenting and as such are not usable as tests until assembled into the binary information tables for operation by the second part of the software system.

The second part is a group of control programs devoted to interpretation and handling of SRL demands and necessary utility functions.

In brief these demands and functions are:

1. Parameterization.
2. Message handling.
3. Interrupt control.
4. Memory allocation.
5. Register allocation.
6. Device control, multiple or single.
7. Channel control, multiple or single.

■ Multi-Processor PTS

PTS is fully reentrant and, as such, can sustain a large number of peripheral test activities concurrrently. Few common data areas exist in PTS; however, those which do exist are protected by test and set interlocks.

Offline PTS adjusts automatically to any possible system configuration. With offline PTS, a much greater control over the active configuration exists than is possible under the Operating System online.

As an example: low conflict mode (storage port conflict), maximum conflict mode, or roll mode (PTS I-bank code, the peripheral activity buffers and SRLs are rotated through storage) may be selected. PTS may also be dynamically locked out of selected areas of storage.

■ Offline to Online Compatibility

PTS is made up of many program elements. Most are the same whether they are incorporated into an online or an offline PTS program. The functional characteristics of the device tests are the same. The PTS worker level source elements are unchanged, however, they must be reassembled with an offline or online flag set in one of the PTS Procedure Definition Processor (PDP) elements which then will generate unique relocatable binary elements. The changes caused in PTS worker level code are, however, not very significant. The most significant difference between online and offline PTS is in the Executive level code. Offline PTS is collected as a single program. This program has the capability to perform test sequences on all peripheral devices in the system. As a single program, with all element interconnection accomplished by the collector prior to generation of a boot tape, PTS has greatly reduced the Executive level work. Additionally all activities, with the exception of the wait loop, are interrupt activities and are given a first come first served priority. Switching occurs only after an activity releases control. The effect is a simplification of the Executive level code in offline PTS.

■ PTS Hardware Components and System Environment

2.3.2.3.1. Online System Environment

In the online mode of operation, PTS is a device oriented system used for performing corrective or preventive maintenance, and is run under control of the 1100 Series Executive. No system control beyond data path selection is possible. The parameterization for device control, however, has the same syntax as offline PTS.

- Online Hardware

PTS provides an online demand mode or batch mode of operation. To accommodate demand mode, a demand terminal should be available to the Customer Engineer.

- Demand Mode – Remote

Ideally peripheral corrective maintenance should be performed online. Demand terminal operation of PTS is the most convenient for the maintenance personnel, and the least visible to the operations personnel. It is, therefore, the most desirable means of controlling PTS in the online environment.

- Batch Mode

Batch mode operation of PTS is via runs submitted to the operations people onsite. These runs are preparameterized. Changes to the parameters are possible via the operator console. This type of operation is very visible to the operations personnel, and is less flexible and convenient for the user than demand mode.

2.3.2.3.2. Offline System Environment

Offline PTS is intended for use in isolating malfunctions which elude isolation by the Operating System, and as such preclude "graceful degradation" of the system. In this contingency the most important requirement of the test software is to immediately cause all sections of the central complex to begin a testing sequence, and to sustain the test environment operation within the system at as high a rate as possible. In this way a high probability of a "hit" in the malfunctioning area is assured. The objective is to find the failing system element (processor, storage, MMA, etc.) and eliminate it from the system. The system may then be returned to the user less the failing element. PTS has five characteristics that facilitate system malfunction isolation.

1. A boot process which loads the entire program with a capability to test the entire complex.

2. Run streams can be defined prior to generation of the boot tape. These run streams contain parameter statements which establish the desired test environment.

3. Storage roll mode which automatically moves the I– and D–banks of the test activities through all of storage.

4. Storage conflict control where simple parameters to PTS control allow establishing a minimum or a maximum probability of conflict at the storage ports.

5. Storage lock out where blocks of storage may be excluded by parameter from the pool of storage available to PTS. The effect is an intensification of use in the areas still available to PTS.

2.3.3. Total Remote Assistance Center (TRACE)

TRACE is a World Wide Customer Engineering (WWCE) product support system located in Roseville, Minnesota. Its purpose is to provide remote support assistance to various SPERRY UNIVAC Systems and terminal products, including many 1100 Series Systems.

1100 Series Systems supported by TRACE Remote Test Applications include:

1110 System
1100/10 System
1100/20 System
1100/40 System
1100/80 System

Terminals supported by TRACE include the following:

UNISCOPE 100/200
DCT-500 Series (DCT-475, -500, -515, -525)
DCT-1000
UTS-400

2.3.3.1. System Remote Test Applications

The 1110 and 1100/40 Systems can be tested by Trace using either the Maintenance Controller Operation (MCO), or the Remote Console Operation (RCO). The 1100/80 System can be tested by TRACE using Remote Control Routine (RCR/80) and CEMAX Auxiliary Console Operation.

MCO allows complete set/scan control by TRACE via the onsite maintenance controller (M1920). Functions include program load, panel control and execution. Panel data scanned by TRACE can be compared to known good data to aid in isolating a hardware problem.

The MCO application is not available for testing the 1100/20, /10 Systems since these systems do not include the onsite maintenance controller scan/set interface.

RCO enables TRACE specialists to perform all system console functions including keyboard entry and screen monitor. Using RCO, the TRACE specialist can load, execute and monitor the 1110 and 1100/40 Central Complex Diagnostic Test System (CCDTS) and the Peripheral Test Sequencer (PTS) for offline system testing. Remote Console Operation is also possible for testing 1100/20, /10 Systems.

The RCR/80 allows TRACE to mointor and/or control of the 1100/80 System Maintenance Unit's system console. All functions that can be performed at the onsite 1100/80 SMU console can be performed at the TRACE Center. These functional capabilities will allow TRACE to load and execute programs and to have maintenance panel set/scan control.

The CEMAX Auxiliary Console Operation will enable the TRACE Specialist to perform the 1100/80 System console operation including keyboard entry and screen monitor control. Using the CEMAX Auxiliary Console Operation the TRACE Specialist can load, execute and monitor the 1100/80 Control Complex Diagnostic Tests and the Peripheral Test Sequencer (PTS) for offline system testing.

The system console handler software of the Operating System has coding included which allows TRACE RCO to interface with the Operating System of 1100 Series Systems using the T4013 console (1110, 1100/10, 1100/20, 1100/40). This feature, included by a special system generation parameter, is enabled by the onsite system operator to insure system integrity. Once enabled, the feature allows the TRACE specialist to control and monitor execution of the onsite Operating System, including such Online Maintenance (OLM) tests as may be available at the site.

2.3.3.2. Terminal Remote Test Applications

Remote support of terminals via TRACE is normally accomplished using one of the following methods:

Direct Support	Describes the situation in which TRACE specialists or regional specialists are engaged in actual execution and monitor of a terminal test.
Auto--Test	Describes the situation in which Customer Engineers execute and monitor a terminal test from the site of the unit under test.

TRACE terminal test programs are high level functional tests, featuring subtest selection and recycling options.

3. Systems Programming

3.1. LOCAL CODE

3.1.1. Tape Label Site Modification

Tape Labeling Site Modification (TLSM) is a capability provided within the tape labeling system for a site to easily alter the tape labeling logic flow to meet the special requirements of individual sites. TLSM is implemented so that a site can choose the appropriate entry point for their modifications; code a routine, (following entry and exit conventions specified for that entry point and standard EXEC conventions) and insert that routine in an element defined for site modifications. This routine will then become an integral part of the 1100 Series Operating System. The site supplied routine will, in effect, define a previously undefined set of processing logic for a specific decision point within tape labeling processing. Therefore, a site will not be modifying, in the traditional sense, the operating system; instead, it will be extending or overriding logic provided by the Operating System's base level implementation.

3.1.1.1. Description of Site Own Code Interfaces

Within the tape labeling system, in the element AAFIPROC and the proc TLPROC, is a series of labels and values which are used to turn on the code which will link to the location in the element TLSM where the site modifications reside. The labels and their descriptions are in Table 3–1.

The facility for site modification of HDR1, EOF1 and EOV1 labels is designed so a site can bypass all of Operating System processing or a site can modify the validation of any field within the label. This is accomplished by table driving the Operating System validation. Entries in the table can be modified or new entries can be inserted according to the rules specified in the element TLSM.

3.1.1.2. Examples for Possible Use of Site Modifications

- EBCDIC VOL1 labels can be validated by turning on UNLBLD and inserting in the element TLSM site modifications to check for an EBCDIC VOL1 and then validate it.
- Additional meanings for VOL1 accessibility can be implemented by turning on ACCRST and WRTVOL and inserting in the element TLSM, site modifications to validate the VOL1 accessibility, when the VOL1 is read and set the VOL1 accessibility, when the VOL1 is written.

Table 3-1. Tape Labeling System Labels

Label	Description
UNLBLD	Provides facility for site modifications when unlabeled volumes are mounted.
VOL1RD	Provides facility for site modifications when labeled volumes are mounted.
ACCRST	Provides facility for site modification of VOL1 restricted access validation.
RDSYS	Provides facility for site processing UVL labels.
RD1TYP	Provides facility for site modification of HDR1, EOF1, and EOV1 validation.
RD2TYP	Provides facility for site modification of HDR2, EOF2, and EOV2 validation.
WRTVOL	Provides facility for site modification of VOL1 labels.
WRT1TYP	Provides facility for site modification of HDR1, EOF1, and EOV1 labels.
WRT2TYP	Provides facility for site modification of HDR2, EOF2, and EOV2 labels.
ERCHK	Provides facility for site modification of any ER TLBL$ request.
RDERLB	Provides facility for site modification of user labels read via ER TLBL$.
WRTER	Provides facility for site modification of user labels written via ER TLBL$.
ERHDRD	Provides facility for site modification of system header or trailer label information retrieved via ER TLBL$.

- UVL labels can be implemented by turning on RDSYS and WRTVOL and inserting site modifications to validate UVL labels and to write the VOL1 label, and create and write additional UVL labels (the return from the WRTVOL facility will write what ever is in the label area of the tape labeling control buffer as a label).

- Read and Write keys in HDR2 labels can be implemented by turning on RD2TYP and WRT2TP and inserting site modifications for creating HDR2, EOF2, and EOV2 labels with keys and validating the keys when the HDR2, EOF2, and EOV2 labels are read.

- New ER TLBL$ functions can be implemented by turning on ERCHK and inserting site modifications to check for and process new functions.

3.1.1.3. Restrictions on Tape Labeling Site Modifications

The following restrictions are part of the tape labeling design. There are no plans to change these restrictions with future modifications to the tape labeling system.

- System Labels

  Additional system labels (HDR3, EOF3) will be difficult to implement due to the requirement that corresponding system labels must appear in both header and trailer label groups. Therefore, information necessary to create a HDR3 label must be saved to create the EOF3 label and presently there is no space in the PCT tape item to save this type of information.

- 9-Bit ASCII Format

  The tape labeling site modification facility does not apply to 9-bit ASCII format labeled tapes. Control will only be passed to the facility if a new format label tape is read or is to be written.

3.1.2. CCRs

The Operating System provides an interface between the user written Communications Control Routine (CCR) and the 1100 Series symbiont complex. The interface is the Executive Request (ER) RSI$. ER RSI$ allows for both demand and batch run operation.

Volume 7 of the 1100 Series Executive Technical Documentation, (ETD-7), contains extensive information and specification of RSI$ and CCR design.

3.1.3. Adding ERs

The user ER facility allows sites to use Executive Requests to the system without conflicting with additional EXEC ERs using the indexes chosen for use by the site. The ER indexes 04000 to 05777 are reserved for the user ER table for site use.

In order to make use of user ERs the site must set the config tag US$ER non-zero. The ER table entries for user ERs are placed in the element AAERTGP following the tag USER. The format is:

```
ERH     address/function-index,ER-type-bits,access-word-count,  point/attach
        ER-SUP-charge  function-entry-point
```

where:

ERH — is the proc call which generates the two word table entries

address/function-index — is the address of the handler for resident ERs or the Function index for ERs which are handled by EXEC functions

ER-type-bits — describe actions to be taken to setup for execution of the ER code

- R raise user level
- A symbiont ER
- F Function handles this ER
- E EXEC only ER
- P ER may cause PCT expansion
- D deactivate user with general wait

access-word-count	specifies a number of words to be checked as valid user addresses assuming A0 has a user packet address
point/attach	FFA attach, FFP point, on FSTART of function to handle the ER
ER-SUP-charge	the SUP charge for handling this ER
function-entry-point	the entry point within the EXEC function where the code to handle this ER begins

Entries in the user ER table are for the ER indexes 04000 to 05777 in order of entry.

3.1.4. Error Messages

The EXEC has been written so that it will be able to printout meaningful diagnostic message for errors encountered in user programs and for errors detected by processors. This message lookup and printout facility is available to user programs through the use of an Executive Request (ER). The text for these messages is kept in a specially formatted omnibus element named E$ORMSG which resides in SYS$*LIB$. Messages may be easily changed or inserted by remaking this omnibus element. The system also provides a facility for demand users to retrieve register and jump stack printouts when their programs error.

In order to make use of the error message printout facility E$ORMSG must be placed in SYS$*LIB$. This omnibus element can be changed or have new messages added to it through use of the ERRORELT processor. If no omnibus element named E$ORMSG appears in SYS$*LIB$ then the message lookup, printout facility and ER ERRPR$ are logically turned off. E$ORMSG should go into the fast LIB$ file on the boot tape at most sites. The processor ERRORELT should not be placed in SYS$*LIB$ because of its extremely low use.

3.1.4.1. ERRORELT Processor

The ERRORELT processor is used to produce the E$ORMSG element for lookup and printout of error messages by the EXEC.

3.1.4.2. Processor Call

The processor calling sequence for ERRORELT is:

```
@label:file.ERRORELT,options  elt-1,elt-2,elt-3
```

where:

elt-1	specifies the source input element, output element for 'I' option, or both for 'U' option.
elt-2	specifies the output element for the omnibus element in E$ORMSG format.
elt-3	specifies the source output element (not used with 'I' or 'U' options).

3.1.4.3. Options

Processor call options applicable to the ERRORELT processor include all of the standard SIR$ options except the "W" option. Additional options are the "D" option, which causes an octal dump of the element produced, the "N" option, which causes no printout of the source input, and the "C" option, which causes printing of the number of printable characters by ERRFO for each input line.

3.1.4.4. Input Format

Input to the ERRORELT processor is in the form of symbolic card images taken from any symbolic element or the run stream. Input is handled by the source input routine (SIR$) so all standard SIR$ processor call options, except the "W" option, apply. The standard processor call card format is used except that the element name in the relocatable output field is used for the output of the omnibus element created by the processor.

The format of the card image input acceptable to this processor consists of a 6-digit octal number in card columns one to six, an "E" or a "P" in column seven specifying an EXEC type message or a processor type message respectively. The sentence text begins in column eight or anywhere thereafter on the card. Continuation of a sentence to additional cards is specified by columns one to six being blank and column seven containing a plus sign (+) on all additional cards needed to contain the sentence. Additional sentences in a message are specified by cards with columns one to seven being blank on the first card of each additional sentence. Continuation of these sentences is specified as above. Sentences of a message are limited to a maximum of 131 characters, for output purposes, including one blank between words and a blank at the end of the message. Multiple blanks between words on input are reduced to one for output and size determination. A word for use by this processor is defined to be any string of contiguous non-blank characters. Words are limited to 18 characters. Sentences are limited to 57 words, and messages are limited to 62 sentences. Words may not be broken at the end of one card and continued to the next.

The format of the 6-digit octal number appearing on the first card of a message for errors detected by the EXEC consists of three 2-digit fields. The fields are defined to be, from left to right, error type, error code, and external contingency number.

Example:

```
012112  error type:  01     I/O error
        error code:  21     AN ATTEMPT TO REFERENCE AN UNASSIGNED FILE
        contingency number:  12     EMODE
```

For those errors determined by a status bit setting, the bit number replaces the code field.

Example:

```
404300  error type:  40     Facility status
        error code:  43     Bit 35 set, FACILITY REQUEST REJECTED
        contingency number:  00     None
```

The format of the 6-digit octal number on the first card of user messages, those with a 'P' following the message number, is currently undefined except that all codes with 070 to 077 in the third sixth of that half-word are reserved for site use.

Examples:

010276 code available for site use

217352 code reserved for Sperry Univac use

Additional input formats available are, cards with an 'X' in column 7 which cause a page eject and those with a 'C' in column 7 which signify comment cards.

Any errors detected in the input to ERRORELT will cause the entire sentence in which the error occurs to be rejected along with all subsequent sentences in the message. A duplicate error number appearing in the input (121212E and 121212P are not duplicate numbers) causes the second and all subsequent messages with this number to be rejected. The output element is produced even when there are non-fatal errors in the input, however the element will be larger than necessary. The errors should be fixed and the element reconstructed using ERRORELT.

3.1.4.5. Processor Detected Input Errors

AN ILLEGAL CHARACTER APPEARS IN MSG NUMBER

A character other than an octal digit appears in the error number.

COL 7 MUST CONTAIN E OR P ON 1ST MSG LINE

On the card specifying the error number an E or a P must be present in column 7 to indicate whether the message is an EXEC or a USER (Processor) message.

ERROR CLOSING SOURCE

SDFO was unable to close the source input or ER PFI$ gave an error status to SIR$.

ERROR ON PROCESSOR CALL CARD

PREPRO detected an error when reading the control card or setting up PARTBL.

ERROR OPENING SOURCE INPUT

SIR$ detected an error while attempting to setup to read the source input.

ERROR UNABLE TO ENTER ELEMENT IN FILE TOC

A bad status was returned on ER PFI$ while attempting to put the information for E$ORMSG in the file's table of contents.

MSG NUMBER APPEARS ON CONTINUATION LINE

The first six columns are not blank on a card with a plus (+) in column seven indicating a continuation line.

NO SOURCE INPUT

End of file status given on first attempt to read the source input.

NO VALID MESSAGES NO ELEMENT PRODUCED

All messages in the input contain errors.

POSTPR ERROR FREEING FILES

POSTPR reference to CSF$ returned a non-zero status.

SENTENCE LONGER THAN 57 WORDS ENCOUNTERED

Sentences are limited to a maximum of 57 words.

SENTENCE LONGER THAN 131 CHARACTERS ENCOUNTERED

Sentences are limited to a maximum of 131 characters.

UNABLE TO ASSIGN SCRATCH FILE PSF$

ER CSF$ attempt to assign scratch file PSF$ resulted in a facility reject status.

WARNING--SENTENCE LONGER THAN 72 CHARACTERS

A sentence was encountered with more than 72 characters. This is a warning only to help format output for terminals which allow only 72 characters per line.

WORD LONGER THAN 18 CHARACTERS ENCOUNTERED

Words are limited to a maximum of 18 characters.

nnnnnn DUPLICATE ERROR MESSAGE NUMBER

A message of the same type, EXEC or USER, already exists with number nnnnnn.

nnnnnn NO TEXT FOR MESSAGE

No input text exists for message number nnnnnn.

3.2. LIBRARY MODIFICATIONS

3.2.1. Common Banks

3.2.1.1. General

A common bank may be used as an I-bank or a D-bank which can be shared by programs from more than one run concurrently. A common bank does not reside within the absolute elements which use it but is rather a distinct absolute element and resides in SYS$*LIB$. However, any user program can use a common bank in exactly the same way that it uses its own banks; that is, by use of LIJ/LDJ instructions to the bank or by initially basing the bank.

Common banks have two principle applications:

1. Reentrantly coded instructions which can be executed from many runs. These are normally high use subroutines. The purpose of this use of common banks is to prevent having several copies of the same routine concurrently resident; that is, to use main storage more efficiently.

2. Common data tables to be used concurrently by several runs. The purpose is to prevent having several copies of the same data tables concurrently resident.

3.2.1.2. Common Bank Creation and Utilization (General Case)

3.2.1.2.1. Description of the General Case

The general case of programs which use common banks is described as follows:

A collection of relocatable elements, some of whose location counters are to be placed in common banks (because their contents subscribe to one of the aforementioned applications) and the remainder of whose location counters are placed into program banks (i.e., each user has his own copy.) Moreover, the user would like to access all of these banks simply by:

@XQT FILENAME.ELTNAME

or

@FILENAME.ELTNAME

In order to do this, the user may have to build several absolute elements. There must be one absolute element for each common bank and one absolute element which contains all of the program banks. The absolute element which contains all the program banks may be placed in any program file (including SYS$*LIB$). It is this element that is named on the @XQT or processor call. The associated common banks will then be based either because they are initially based (see below) or because an LIJ/LDJ from these banks (or another common bank) bases them. As was mentioned above, there must be one absolute element for each common bank used, as well as one absolute containing all of the program banks. Correspondingly, there must be the same number of @MAPs (collections) – one to build each of the absolute elements involved.

3.2.1.2.2. Explanation

In order to understand the reasoning behind the mechanics of building and using common banked programs, certain functional aspects of the Collector must be discussed. This is the purpose of the present section. This discussion will begin with an example. Suppose that one wants to construct a program with one common bank and, we'll say, 3 program banks. Then, from what has been said, the user must perform 2 collections: one which builds an absolute element containing the 3 program banks, and one which builds an absolute element containing only the common bank. From this, it may seem reasonable to assume that the collection which builds the common bank should specifically include only those location counters to be included in the common bank and similarly for the absolute element containing the program banks.

Unfortunately, there are many reasons why this will not work. In general, we must assume that location counters within the program banks may reference entry points within location counters of the common banks and conversely. Therefore, the location counters of the program banks must be available during the collection of the common bank so that the Collector will be able to resolve any external references from the common bank to entry points within the program banks. But, of course

this program bank code must not actually be placed in the absolute element containing the common bank. It must be stripped off ("V" option).

The procedure established in the following subsection will prescribe that the Collector directives for both of these collections (one for the common bank element and one for the program bank element) will be exactly the same and will use all banks involved, with the single exception of the fact that a "V" option will be applied to I-bank or D-bank directives whose code should not actually be contained in the resulting abosolute element. (The "V" option on a bank directive indicates that the Collector is to use the location counters of the bank to satisfy external references to the other banks, but is not to place the code from that bank into the absolute element being produced.)

■ Building Common Bank Programs

A common banked program may possess multiple common banks and program banks. There are three major steps to building a common banked program.

A. One collection for each common bank element.

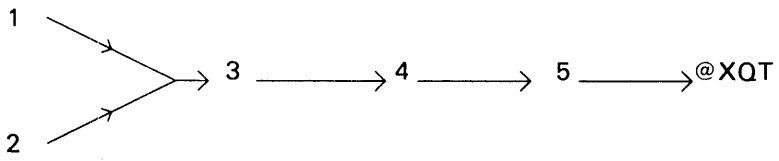
B. Insertion of the common bank(s) into the Operating System.

C. A collection for the program banks element.

Effecting these major steps is accomplished in five phases as follows:

Step	Phase
A	1. Collections of the common bank elements
B	2. Creation of a new CERU$ element
B	3. Placement of the common banks and the new CERU$ elements into the LIB tape.
B	4. SYSGEN & Bootstrap to insert the common bank into the system.
C	5. Collection of the element containing the program banks.

The following is a critical path schedule for the above five phases.



The following paragraphs detail these five phases. The term 'X-bank' means "I-bank or D-bank".

■ Collections of the Common Bank Elements

For each common bank there must be a separate collection. The Collector directive for each of these collections must be exactly the same, with one exception – placement of the "V" option on X-bank directive. Every bank in the program, including each common bank and each program bank, must be represented in each of these collections.

The only difference between these collections should be that

1. For each collection, every X-bank directive should have the 'V' option except the bank whose code is to become the absolute element being produced by the collection.

2. Within the common banks being created, any reference to the BDIs of other common banks in the program must be hard-coded and equal to the BDI values of the common banks which will be established during Phase 2 and during Phase 4 of the reentrant SGS. (This is true because SYS$*RLIB$ does not yet contain the BDIs of these common banks and because the Collector would otherwise assign a BDI value to these references based on the nature and relative positions of the X-bank directives of the other common banks directives within the collection. If at least one of these common banks will be referenced by at least one of the program banks, there is no problem with PSR bit D19 being set.) It is also suggested that the bank names on any 'V' option X-bank directives for common banks be dummy names.

3. The RO field of each @MAP statement should contain a unique element name. Generally, this should be the element name desired for the common bank element once it is placed in SYS$*LIB$.

CERU$ Element

CERU$/*version* (*version* is the current version) is a relocatable element in SYS$*RLIB$ which contains the BDIs of all common banks. When a relocatable element references a common bank at collection time, its BDI will normally not be defined to the collector. This causes the Collector to search RLIB$ for a definition. If the BDI reference is resolved in CERU$, the Collector indicates in the absolute element that bit D19 in the program's PSR should be set. This gives the program permission to use common banks. Therefore:

1. The name of any common bank must appear in SYS$*RLIB$.CERU$/*version*.

2. Common bank BDIs must not be hard coded in program banks. They must be referenced by their bank name as defined in CERU$/*version*.

CERU$ must be updated so as to include the names of new common banks. However, it is not a symbolic element. So it must be created from scratch. Any old information must be retained and new common bank names entered. Now, each line in the symbolic CERU$ is a EQU directive of the form:

```
Bankname*  EQU  BDI
```

For Example,

```
ASM$*  EQU  0400005
```

The BDI must match the BDI named on the REPROG SGS entry for the bank at system generation. Common bank BDIs must:

1. Have bit 17 set to 1.

2. Be larger than 0.

3. Be smaller than or equal to the CONFIG tag MAXBDI.

4. Must be unique.

The tag MAXBDI may need to be readjusted. Common Bank BDIs should be contiguously numbered starting with 0400001. (Otherwise, space is wasted in the EXEC's BDT.) Now, one must write and assemble the symbolic element CERU$. To get the BDIs perform the following:

@LIST,R SYS$*RLIB$.CERU$/*version*

Where *version* is current version of CERU$ in the system.

EQUs for the new common bank must be added.

■ New Common Banks and CERU$ to LIB Tape

The LIB Tape has 4 files:

1. LIB$1 (Fast LIB$)
2. LIB$2 (Slow LIB$)
3. RUN$ (System Canned Runs)
4. RLIB$1

The newly created common banks must be copied into either file 1 or 2 of the LIB tape.

The newly created CERU$ must be copied into file 4 of the LIB tape.

■ SYSGEN and Bootstrap to Insert the Common Banks

Each common bank in the system must be represented on a REPROG SGS at system generation, including the new ones. The format of the REPROG SGS is:

REPROG BANKS,Options ARE 'ELTN',BDI 'ELTN',BDI ...

'ELTN' is the element name in LIB$ of a common bank. 'BDI' is the BDI of the common bank.

This BDI must match the BDI for this bank within the new CERU$ element which was placed on the LIB tape during Phase 3.

Since the new CERU$ is not yet in SYS$*RLIB$ during this system generation, new common bank BDIs should be hard coded on the REPROG SGS for this system generation.

Example:

REPROG BANKS,S+RE+W ARE 'NEWCB',0400016

Once the system is built, it should be booted prior to Phase 5. (Otherwise the Collector may not set PSR Bit D19.)

■ Collection of the Program Banks Element

Finally, one collection is required for the absolute element containing the (non-common) program banks. The collection must contain exactly the same MAP directives as the collection(s) which built the common bank(s) (Phase 1). Any common bank(s) and all the program banks must be represented; the only difference between these collections will be that:

1. The X-bank directives of all the common banks should have the "V" option. The X-bank directives for the program banks should not have the 'V' option.

2. The bank names on the X-bank directives for the common banks must be dummy names; otherwise the Collector will not get the BDIs from CERU$.

3. The RO field of the @MAP statement should place the program under the element name by which it should be executed.

4. It is important to note that each common bank that is to be initially based should have a second X-bank directive possessing the "X" option. This X-bank directive must have an "M" or a "U" option, as well. The proper bankname of the common bank should be placed on the "X" option X-bank directive. Such an "X" option X-bank directive should not have any IN, SEG or FORM directives between it and the following X-bank directive.

Once the collection has finished, the program can be executed by referencing the name in the RO field of the MAP directive on a @XQT statement or a processor call statement.

3.2.1.2.3. Example of the General Case

Suppose that a common banked program 'THRIFTY' has six banks, two of which are to be common banks and four of which are to be program banks. One of the common banks, call 'CDATA', contains a common data look-up table. The other common bank, call 'CSUBS', contains all reentrantly coded routines that exist in the program. The program banks contain only data that is peculiar to individual users and routines that are not or cannot be reentrant (such as code generated by compilers that cannot be generated to be reentrant in nature.) (Note that the ASCII COBOL Compiler does generate reentrant code.) Finally, assume that 'CSUBS' is to be initially based on PSRU.

■ Collection of the Common Banks Elements

The following collections build the common banks.

```
@ASG,C      CAPTURE.,F
@MAP,IL     CAPTURE.  CDATA$
D-BANK      CDATA
    IN
    •
    •
    •
I-BANK,V    CSUBSDUMMY
    IN
    •
    •
    •
I-BANK,V    11
    IN
    •
    •
    •
I-BANK,V    12
    IN
    •
    •
    •
```

```
I-BANK,V     13
     IN
     •
     •
     •
D-BANK,CV    CONTROL  BANK
     IN
     •
     •
     •
END


@MAP,IL      ,CAPTURE.CSUB$
D-BANK,V     CDATADUMMY
     IN      -----
     •
     •
     •
I-BANK       CSUBS
     IN      -----
     •
     •
     •
I-BANK,V     11
     IN      -----
     •
     •
     •
I-BANK,V     12
     IN      -----
     •
     •
     •
I-BANK,V     13
     IN      -----
     •
     •
     •
D-BANK,CV    CONTROL  BANK
     IN      -----
     •
     •
     •
END
```

■ Example of Creating a CERU$ Element

Assuming that we have no symbolics of CERU$ available, we must determine what common banks are already there so as to preserve them. The information may be obtained by the following:

```
@LIST,R   SYS$*RLIB$.CERU$
        or
@MAP,IL
    IN SYS$*RLIB$.CERU$
```

(Ignore the $MAIN$ entry in the resulting listing.)

Having obtained this information we create a new CERU$ which preserves the old information and adds the information regarding our 2 new common banks.

```
@RUN
@ASG,A      CAPTURE
@ASM,IS     CAPTURE.CERU$,CAPTURE.CERU$
ASM$*       EQU     0400005          The old information
ED$V*       EQU     0400007
RMATH$*     EQU     0400001
CMATH$*     EQU     0400003
NIOCB$*     EQU     0400004
DMATH$*     EQU     0400002
NWBCB$*     EQU     0400006
CSUBS*      EQU     0400010          The new CB Information
CDATA*      EQU     0400011
END
```

■ Example of Common Banks and New CERU$ to LIB Tape

```
@ASG,T      OLDLIB.,T.XXX
@ASG,T      LIB$1,F  FAST LIB
@ASG,A      CAPTURE
@COPIN      OLDLIB.,LIB$1.           Get old LIB$1
@COPY,A     CAPTURE.DATA$,LIB$1.     Copy CBs to LIB$
@COPY,A     CAPTURE.CSUBS$,LIB$1
@ASG,T      NEWLIB., U9,Y
@COPOUT     LIB$1.,NEWLIB.           Put new LIB$1 on LIB tape
@COPY,M     OLDLIB., NEW LIB.,2      Put LIB$2 and RUN$ on new LIB
                                     tape
@FREE       LIB$1.
@ASG,T      RLIB$1.,F
@COPIN      OLDLIB.,RLIB$1.          Get LIB$1
@COPY,R     CAPTURE.CERU$,RLIB$1.    Put new CERU$ in RLIB
@COPOUT     RLIB$1.,NEWLIB.          Put new LIB$1 on new LIB tape
@COPY,M     OLDLIB.,NEWLIB.          Put LIB$2 on new LIB tape
```

■ SYSGEN and Bootstrap to Insert Common Bank

A SYSGEN must to done to insert the common bank into the system. Present must be a MAXBDI SGS to assure that the EXEC's Bank Descriptor Table (BDT) is large enough, and a reentrant SGS representing each common bank, including the new common banks. In our examples:

MAXBDI

011

Must be present, since our largest EXEC (Common) BDI is 0400011,reentrant SGSs must be present for the new ones, as well as reentrant SGS's for the old common banks. If two common banks have the same properties (options) they can be placed on the same REPROG SGS.

REPROG banks,D+PE+W are 'CSUBS$',0400010 'CDATA$',0400011

This system must be booted prior to performing Phase 5.

■ Collecting the Program Banks Element

It is assumed that the new system is booted while performing the following collection. Recall that the map directives for this collection are the same as for the collection of CSUB and CDATA except that:

1. The X-bank directives for CSUB and CDATA will have the 'V' option, while the others will not.

2. The X-bank directives for CDATA and CSUBS will have dummy names so that references to them from the program banks will be resolved from the new SYS$*RLIB$.CERU$.

3. The common bank 'CSUBS' will have an extra X-bank directive possessing the 'X' option since it is to be initially based when initial loading of the program. Its X-bank directive must also have an 'M' or 'U' option (1110 only) to indicate the PSR on which it is to be initially based. In our case, 'CSUBS' is to be initially based on PSRU. No IN, SEG or FORM directives will follow this I-bank directive. This X-bank directive should contain the correct name: 'CSUBS'. However, the Collector will resolve references to the tag 'CSUBS' from CERU$ (unless otherwise defined in the collection.)

The following will produce the user absolute program:

```
@ASG,C      PF.,F
@MAP,IL     ,PF.THIRFTY                    CREATES 'THIRFTY'
I-BANK,XU   CSUBS
D-BANK,V    DUMMYCDATA
    IN      -----
    .
    .
    .
I-BANK,V    DUMMYCSUBS
    IN      -----
    .
    .
    .
I-BANK,M    11
    IN      -----
    .
    .
    .
I-BANK      12
    IN      -----
I-BANK,D    13
    IN      -----
```

```
      •
      •
      •
D-BANK,CM   CONTROL BANK
    IN      -----
      •
      •
      •
```

The following will execute the common banked program PF.THRIFTY. The program banks 11 and D1 will be initially loaded and based on the main PSR. The common bank CSUBS will be initially loaded and based on BI of the utility PSR. The program bank I2 will be initially loaded (it is static) but will not be initially based:

```
@XQT        PF.THIRFTY.
```

If PF.THIRFTY were written with the expectation that it were to be called as a "processor", the following would successfully do the same as described above:

```
@PF.THRIFTY
```

3.2.1.2.4. Applications

Any program which is expected to be used concurrently by more than one run can and should be common banked if:

1. it contains reentrantly generated code;

2. it contains read-only data; or

3. it has a need for each of its several concurrent users to communicate with each other.

At any point in time, the main storage 'saved' by this application is a function of the size of the common banks and the number of concurrent users of each. The saving is particularly high when any of the common banks are static. This application is particularly useful for ASCII COBOL and ASCII FORTRAN generated programs since both compilers generate reentrant code.

■ Some Collection Problems

As has been indicated, several collections may be necessary to build a common banked program: one collection for each common bank, and one collection for the user banks.

It is important to note that the order in which the Collector places the location counters of included elements must be exactly the same in all collections involved.

If the contents of a program file changes between one of the collections and another, the order of element inclusion may change. This may cause the Collector to resolve references with different map addresses.

This problem may occur when naming a filename on an IN directive or when "LIBing" a file.

The problem can generally be avoided by naming each individual element in the collection on an IN directive. It is also important that all X-bank directives in these collection have their physical order preserved across these collections in order to preserve their BDI assignments. Also, a special problem may arise in the case where there is exactly one common and one user bank with the common bank

being initially based. Because of initial basing, collection of the user bank element will have three X-bank directives (one with the "X" option), whereas the common bank collection will have only two X-bank directives. This may cause the Collector to place location counters of implicitly included elements differently for these collections. The problem is solved by placing a dummy X-bank directive in the common bank collection in the same position as the "X" option X-bank directive. This dummy should not have the "X" option and should not be initially based or the control bank. This bank should be empty. This problem could also be solved by explicitly including all elements in both collections. The latter solution is, as above, recommended.

3.2.1.3. Special Cases

- Cases When Phases 1 Through 4 Can Be Omitted

Often, common banks make no reference to user banks in a common banked program. This occurs when a common bank contains only subroutines. (For example, use of the FORTRAN Common Bank Libraries.) This is also the case for common data banks. When common banks make no references to user banks, then there is no need to rebuild them for use by separate user programs. Also, there is no need to replace them in the system. Therefore Phases 1 thru 4 (see 3.2.1.2.2) may be omitted. During Phase 5 (collection of the element containing the program banks), one of two procedures may be followed. The first is the standard procedure, The second is a method of "cheating".

- The Standard Method

In the standard method, all relocatable elements included in the common banks or the user banks should be included. X-bank directives for all banks, common and user, should be present. X-bank directives for the common banks should contain the "V" option in order to strip off the code in the absolute element.

- Reentrant (Common Bank) Processors.

A reentrant common bank processor is an absolute program consisting of two or more absolute elements, at least one of which is a common bank, but all of which reside in LIB$.

Most reentrant common bank processors consist of one I-bank and one D-bank, with the I-bank being a common bank and the D-bank not common. The starting address is normally in the I-bank.

The common bank assembler has this property. If three runs are executing the common bank assembler concurrently, each will have its own copy of the assembler's D-bank, but will share a single copy of the Assembler common I-bank.

There is, of course, no restriction on the number of banks in a common bank processor. It may possess any number of common banks (within system limits) and as many as 250 program banks (plus PCT).

There is a facility available to common banks which is designed to offer protection to reentrant common bank processors. This feature is called Guaranteed Entry Point (GEP). A common bank can be given the GEP property by specifying the "G" option on its reentrant SGS at SYSGEN time. A GEP common bank has the property that it can only be entered at the programs starting address.

Any common bank may be given the GEP property, but the feature is designed primarily for reentrant common bank processors. Another property of a GEP common bank is that data within it can be accessed only by instructions within it. Thus, the GEP feature may be used to provide a certain method of securing data within the bank.

3.2.1.4. Testing Common Banks

There are two levels for testing common banks. The first is functional testing to checkout the logic and coding of the banks (common and user) to see if they perform the required tasks. The second is reentrant testing to determine if the common banks are coded reentrantly. Considerations for testing common banks include the assumption that one would like to test the banks prior to inserting them into the system.

3.2.1.4.1. Functional Testing

To test common banks functionally, simply collect them together with a set of user banks, producing one absolute element. All banks should be represented by an X-bank directive and none should have its code stripped (no "V" option).

The program can then be executed with test data.

3.2.1.4.2. Reentrant Testing

To test a common banks reentrancy properties, write a multiactivity test driver program. Each activity should base the common bank and use it. There should be times when more than one activity should have the common banks based concurrently.

3.2.1.5. Multibanking

3.2.1.5.1. Multibanking Properties and Restrictions

Some properties and restrictions of multibanking are:

- It allows for program of unlimited size without requiring overlay segmentation.
- It provide the ability to bring into main storage infrequently used code without overlaying.
- It provides for a program growing or shrinking without the MCORE$/ LCORE$ requests.
- It provides for partitioning any program into a maximum of 250 user banks. Therefore, a program may have several smaller banks rather than a few large banks. This allows the EXEC greater flexibility when placing it in main storage.
- The EXEC can swap out dynamic banks not being used, rather than entire programs.
- The programmer has a certain control over which banks can be swapped, and when.
- The programmer has control over placement of banks between primary and extended storage.
- Common banks can be created to be used by several runs concurrently.
- There is a system maximum of 4096 common banks. All runs have access to all common banks.
- Any bank may be defined as I or D.
- A bank may be defined as:

Dynamic: is swappable – independent of other program banks when not based by program.
Static: must be resident for any program activities to be active.

- Any bank may be segmented.
- The segmentation structure of any bank may be independent of other banks.
- In order for a bank to be used, it must be based.
- On 1110 and 1100/40, up to two I–banks and two D–banks may be based simultaneously.
- On an 1108, 1106, 1100/10, and 1100/20, one I–bank and one D–bank may be based simultaneously.
- If a resident bank is defined as dynamic and it is not currently based, then it is individually swappable.
- An individual location counter may be isolated and placed into a selected segment of a selected bank.
- If an element is included in a collection, all of its location counters are placed somewhere in the collection.
- Several copies of an element may exist in a single collection.
- Some Occasions that Call for Multibanking

The following is a list of some occasions that call for multibanking.

1. Very large programs.
2. Programs with large data arrays (blocks).
3. Programs which use infrequently used rountines.
4. Programs which use infrequently accessed data areas.
5. Program primary/extended storage partitioning/placement is desired.
6. Programs which are designed in such a way as to perform a sequence of "pieces" of work. Each "piece" of work could be self–contained in a family of banks, all of which are to be based together.
7. Mixes of runs which use common data areas.
8. Mixes of runs which use common reentrantly coded routines.

3.2.1.6. Non–Configured Common Banks

3.2.1.6.1. General

Non-configured Common Banks allow for more convenient usage of common bank capability. These are banks which are really common in the sense of being saved on swapfile and being shared by all users. These banks need not be configured in the system via REPROG cards, nor need they reside in LIB$.

For some applications, Configured Common Banks are optimal; for applications where Configured Common Banks have been difficult to use, Non-Configured Common Banks offer an alternative.

3.2.1.6.2. Establishing a Non-Configured Common Bank

There are two different types of Non-Configured Common Banks (NCCB). One, called a Stand Alone NCCB, has only one bank within its absolute element. The symbolic BDI used to reference a Stand Alone NCCB is the name of the absolute element which contains the Stand Alone NCCB. The program which uses a Stand Alone NCCB has no special restrictions. The absolute element which contains the Stand Alone NCCB must be in a registered file.

The other type of Non-Configured Common Bank is called an Integrated NCCB. A program which uses an Integrated NCCB must be executed from a registered file. The absolute element which is executed contains the actual code or data which is to be used as an Integrated NCCB.

See Tables 3-2 and 3-3 for description of those tables pertinent to NCCBs.

With Non-Configured Common Banks, there is no longer a need for a unique BDI value, however, a Non-Configured Common bank does require a unique symbolic name.

For this reason, and for reasons of internal resources, it is desirable to restrict generation of Non-Configured Common Banks to elements coming from registered files. There is no restriction that Common Banks come from LIB$ although LIB$ is always a registered file for generation of Non-Configured Common Banks. Any number of additional files can be configured at system generation time. These files are configured by using a CBANKF statement which names the registered files. The format of the CBANKF statement is:

```
CBANKF filename [,filename,. . .]
```

The only restrictions on these files is that they have the qualifier SYS$ and that they be catalogued public. Specifically, they need not be available at boot time nor need they be read only. They are not system files, they are user files whose name is known to the system.

The code for a Stand Alone NCCB is generated by placing an absolute element of the desired name in any of the registered files. This absolute element must contain a single bank. A Stand Alone NCCB is referenced by a program which contains a void bank (length = 0) of the desired name which has the 'S' option on the bank card. This program can be executed from any file. The first reference to a Stand Alone NCCB will actually generate the bank. Subsequent references to the bank will use the single system copy of the bank taken from main storage or loaded from swapfile.

The code for an Integrated NCCB is generated by putting the 'S' option on the bank card of the bank which will be an Integrated NCCB. The bank name supplied to the Collector for the bank with the 'S' option will become the name of the Integrated NCCB. This must not be a void bank. The absolute element which contains the Integrated NCCB may contain any number of (1) real program banks, (2) references to Stand Alone NCCBs, (3) references to Configured Common Banks and (4) other Integrated NCCBs. The name given to an Integrated NCCB must be unique among all Non-Configured Common Banks including Stand Alone NCCBs. An Integrated NCCB defined in this manner is generated on the first execution of this program from any of the registered Non-Configured Common

Bank files. Subsequent executions of this absolute element will use the single system copy of the Integrated NCCB taken from main storage or swapfile.

Guaranteed Entry, Test and Set Queueing, or Contingency Registration can be specified for a Non-Configured Common Bank by putting the appropriate option on the bank card at collection time. For Stand Alone NCCBs, the collection which generated the single bank absolute element must have specified the appropriate options.

3.2.1.6.3. Use of Non-Configured Common Banks

The primary reason for implementing Non-Configured Common Banks is to make common banks easier to use. A Configured Common Bank needs to be in LIB$; it needs a system-wide BDI. Without Non-Configured Common banks, anyone wishing to produce a common bank needs to debug it as a user program, put the BDI in RLIB$, recollect it, have it put on the boot tape in LIB$, and have it configured in the system. If there are problems (its re-entrancy could not be tested until it was in LIB$), it is necessary to rebuild the boot tape and reboot the system. Since there is a very limited number (4095) of common BDIs and only 20 are reserved for site usage, it is to be expected that conflicts with BDI values will occur whenever a site defines BDIs. Even if a site is careful to avoid duplication, there is the problem of the BDI being duplicated in software from Sperry Univac or from another site.

If a BDI is to be changed, it must be modified in RLIB$, which is again a change in the boot tape, and then every program which references that BDI must be recollected.

With Non-Configured Common Banks, many difficulties are alleviated. If a programmer or group of programmers will be producing common banks, they need to have one or more files configured into the system at system generation time. These files need not be on the boot tape, in fact they need not even exist. All that is required is a name. The qualifier of the file must be SYS$. When the time comes to debug the common bank, a unique name must be selected (there are 38^{12} possibilities.) If integrated common banks are to be used, each common bank must have the 'S' option on the bank card. The program can be tested without common banking by executing it in a non-registered file. When it is time to make it a Common Bank, it is copied, without recollection into a registered file. All executions from this file now use the defined common banks. If the Non-Configured Common Bank is to be changed, it is updated in the registered file, with no need to modify the boot tape. Then a common bank reload is done. After reload, all executions of the program use the new version of the Non-Configured Common Banks. This change can be done many times with no need to interface with site management. If it is desired, the program with the Non-Configured Common Banks can be put in LIB$. At this time, it is necessary to rebuild the boot tape, but then, the program has been thoroughly tested. There is no numeric BDI associated with the Non-Configured Common bank. There is no executive data space allocated until the bank is in use. With so many possible symbolic BDIs, it should be an easy problem to avoid duplication. Sperry Univac has a standard for BDI names, so these are easy to avoid. At no time is it necessary to configure a Non-Configured Common Bank into the system.

Use of Stand Alone NCCBs is very similar to that for Integrated NCCBs. For debug purposes prior to using Non-Configured Common Banks, the bank can be collected with the programs. When testing is to be done as a common bank, a void bank with the 'S' option will replace the test common bank. The Stand Alone NCCB will be placed as a single bank absolute element of the right name into one of the registered files. If a program uses many Stand Alone NCCBs, and one of them requires updating, only that one bank needs to be collected with the program for debug purposes.

NCCBs make generating, debugging and updating of common banks very much easier. Non-Configured Common Banks are real shared banks, just as common as Configured Common Banks.

The Common Bank Name Table (CBNTBL) and the Common Bank Element Table (CBENTB) each consist of one control word and a maximum of ten 3-word entries. The entry format is shown in Table 3-2 and is described below.

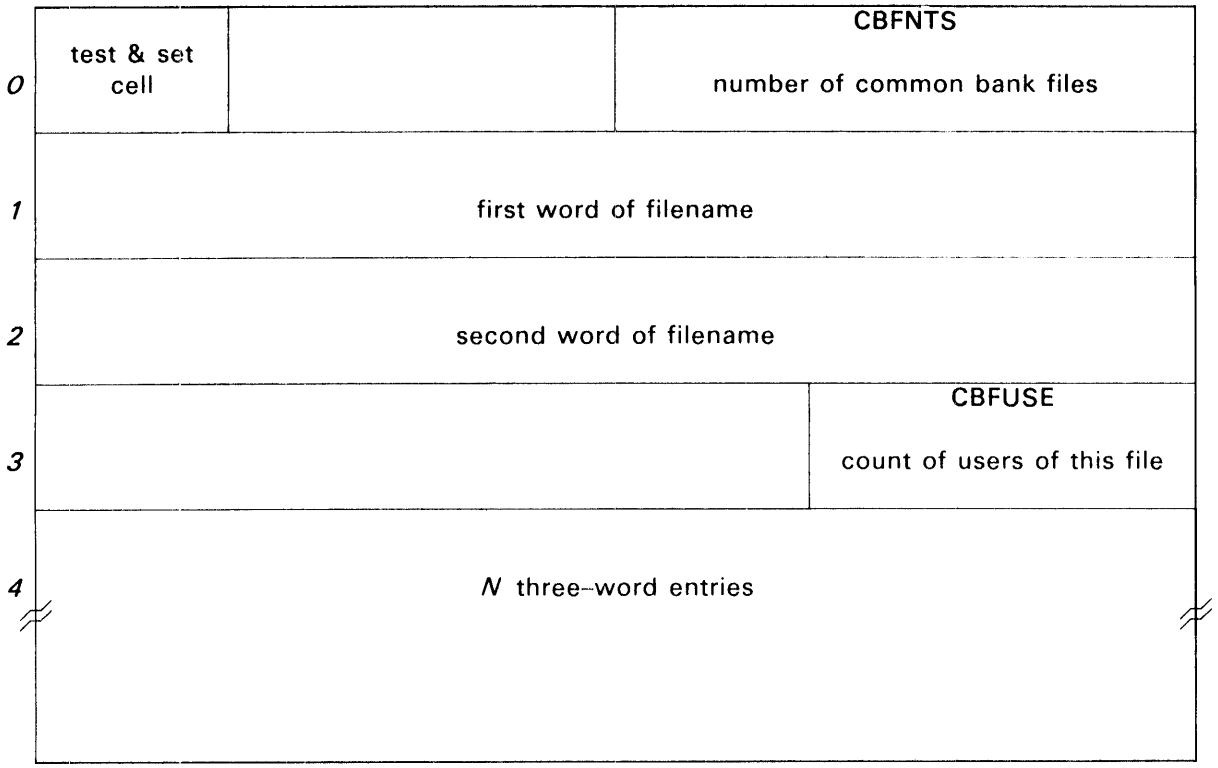
Table 3-2. Common Bank Name and Element Table (CBNTBL and CBENTB)

0		CBNTS3 number of entries in table	CBNTLK link to next table if this one is full
CBNTN1 *1*	first word of bank or element name		
CBNTN2 *2*	second word of bank or element name		
3	Unused	CBBITS *CBNTBL only* info required for NCCBs	CBNTBC bank control packet address
4	up to 012 three-word entries		
037			

The maximum number of 3 word entries per table is 012 and the write protect (value for CBBITS — CBNTBL only) is 010.

The Common Bank Filename Table (CBFNT) consists of a control word followed by 3-word entries. The entry format is shown in Table 3-3.

Table 3-3. Common Bank Filename Table (CBFNT)



where:

CBFNTS = 3 is the size of each entry.

4. System Initialization and Recovery

4.1. BOOT PROCESS PHILOSOPHY

The bootstrap elements of the 1100 Series Executive (EXEC) contain the routines which:

- Perform the initial load from tape.
- Write the EXEC on the system drum/disk
- Load and execute the EXEC from the system drum/disk.
- Take a panic dump in the event of a failure.
- Perform automatic recovery upon completion of the panic dump.

Several parameters are defined in the element CONFIG to provide the flexibility required for various sites. These parameters and their meanings are:

- AUTORECOV

  This parameter is used to specify the action taken when an EXEC error occurs. The options are autorecovery or a zero stop.

- DCBOOT

  Allows booting to disk if ON and prevents booting to disk if OFF.

The EXEC DEVICES statement allows site flexibility in defining the default boot tape device, dump tape device and bootstrap/initialization console device.

4.2. TYPES OF BOOTS

Bootstrap is the process of loading into storage and starting the 1100 Series Operating System. The process is controlled by the bootstrap and initialization elements of the EXEC. There are two types of system boots, initial boot and recovery boot.

1. Initial Boot

   Initial Boot can only be done from tape. The important features are:

   a. The mass storage file area is completely initialized.

   b. The systems libraries are loaded from the tape.

2. Recovery Boot

   Recovery Boots can be done from tape or mass storage. Recovery Boots can be initiated either manual or automatic.

   a. Manual recovery

      Manual recovery can be done from tape or mass storage. The manual recovery is initiated by the operator. It never initializes the mass storage file area except when the file description on a particular device is destroyed. The libraries may be reloaded if the recovery boot is from tape.

   b. Automatic recovery

      Either hardware or software can initiate an auto-recovery boot. An auto-recovery boot is similar to the manual mass storage recovery boot except that a dump of storage may be written to the mass storage EXEC file. The dump can be saved for later analysis.

4.2.1. Tape Boot

The major events of a tape boot are:

1. The hardware loads the bootstrap code from the tape.
2. The bootstrap moves the Operating System from tape to the designated mass storage device.
3. Load the resident portion of the Operating System into main storage from the mass storage device.
4. Start the initialization portion of the Operating System.
5. The initialization section initializes or recovers the mass storage file structure.
6. Initialize the Operating System.
7. Start a runstream that may load the system libraries.

4.3. TYPES OF DUMPS

Dumps are a printed copy of main storage, usually at the time of system errors. Dumps of the Operating System can be taken offline, online or automatic.

1. Offline Dumps

   The bootstrap element contains code to write a dump of main storage directly to tape.

2. Online Dumps

   The Operating System partially suspends itself, writes a dump of storage to a system file, and then continues without rebooting. The dump is transferred to tape or another file.

3. Automatic Dump

   The Operating System or hardware detects a condition to cause the system to take a dump and reboot itself. This dump of main storage is placed in the system file (SWAP$FILE), and later to tape or another mass storage file.

All dumps are subsequently converted from raw format, analyzed and printed by the FLIT or PANIC processor. Features in the printout include a dump of the Operating System's D-bank or a dump of all of storage. Also included in the formatted output are:

- EXPOOL Traces
- I/O Traces
- Function Loader Traces
- Formatted Register and Processor State Register at time of error.
- Cross reference listing of externalized tags.
- Map of the Operating System
- Map of Main Storage.
- Quick analysis of the condition which detected the error.

The SPERRY UNIVAC 1100 Series Executive System Operator Reference, UP-7928 (current version) contains detailed descriptions of dumping procedures.

4.4. OPERATOR RESPONSIBILITIES

4.4.1. Protection of System File Keys

Within the Operating System, three system files (Summary Account Privilege File, System File, and TSS System File) are protected with 'keys' which require special handling. The Summary Account File is protected with a master account number which is solicited from the console operator during the initial boot process. The System File named 'DLOC$' is protected with a read and a write key. Protection for the TSS System File is provided by a master key provided by the site manager when the file is initially created. The following procedures are recommended as a means of protecting unauthorized access to these systems files.

4.4.1.1. Summary Account File Protection

With the QUOTA system a master account and user-id are used to signify the privilege of accessing the Summary Account File. As a result, these appear in the system log, thereby allowing individuals having access to the log to retrieve the master account and user-id and access the Summary Account File as well. To avoid unauthorized access to this file, it is recommended that the QUIP 'CHANGE' command to be used following each privileged QUIP execution or initial boot to modify the master account and user-id. In this way, the current values will not be available to those processing the system log.

4.4.1.2. TSS File Protection

The TSS File is protected with a master key. This master key is provided by the site manager when the file is initially created via the LOCK$ (see 8.2.2.1). The recommended procedure when performing initial boots is to insure that no other batch or demand run is allowed to execute before the TSS File is created. Failure to do so could result in a user created TSS File via a batch or demand run and thereby removing control of the TSS File from the site manager since the site manager would not have access to the new user created master key.

4.4.1.3. 'DLOC$' File Protection.

The 'DLOC$' file is protected with a read key and a write key. These keys appear in listings of the EXEC since one of the EXEC elements must assign this file during system initialization. Since these keys appear in the EXEC listings, it is recommended that they be changed after the 'SYS FIN' message of the boot process but before any other batch or demand runs are allowed to run. The '@CHG' command can be used to change the 'DLOC$' keys.

5. System Tuning and Testing

5.1. SPERRY UNIVAC SOFTWARE INSTRUMENTATION PACKAGE (SIP)

5.1.1. Introduction

This section contains a description of the current status of the 1100 Series Software Instrumentation Package (SIP). It is expected that the SIP will be a continually evolving effort as dictated by hardware and software enhancements to the 1100 Series. Included in the document are a functional description of the SIP, the system generation requirements, the SIP–operator interface and description of the data reduction programs. For complete operating information see the current SIP Software Release Documentation.

5.1.1.1. Types of Software Monitoring

Software monitoring can be divided into two classes based upon the desired objectives of the system analyst. The classes are run–oriented accounting information and system–oriented statistics.

The run–oriented information is similar to that currently available in the log files. These statistics provide summary activity profiles for individual runs and the Executive. This information allows the construction of various program utilization statistics, distributions of run CPU time, breakdowns of run characteristics by remote site, etc. Most of the information required in this class is available in the log files. It is up to the user to provide the data reduction programs which will present the information in a format meaningful to the system analyst. The run–oriented information is most useful in aiding the system analyst in tailoring the job mix in an attempt to optimize system throughput.

Another means of optimizing system throughput is by tailoring the system, both hardware (configuration) and software. It is in this area that the system–oriented statistics are used. These statistics allow the system analyst to:

1. Gain insight into the factors which significantly affect system performance.
2. Aid in the detection and elimination of performance bottlenecks in existing systems.
3. Determine the effect of configuration changes upon system performance.
4. Gather data necessary for the projection of performance increments resulting from improved peripherals.

It is with these system optimization goals in mind that the software instrumentation package (SIP) has been developed.

5.1.1.2. Philosophy of Implementation

In measuring the performance of a system, a primary goal must be the minimization of the effect of the measurements upon the operation of the system. Throughout the development of the SIP, this goal has been a primary guideline. In the following discussion, this guideline will be frequently referenced.

This SIP is event oriented. In other words, the occurrence of an event (incrementation of a queue, change of a storage map, etc.) causes the associated statistics to be accumulated. This technique is in contrast to time oriented statistics which are gathered at periodic intervals and are essentially sampling of states of the system (i.e., periodic sampling of queue sizes, storage bit maps, etc.). The event oriented statistics consist of the number of occurrences of the events, event timings, time of occurrence of the event and, if applicable, the duration of the associated state, and related distributions.

SIP is embedded in the Executive. It consists of strategically placed instructions in a number of elements, the restricted-usage ER SYSBAL$, an additional resident Executive element (SYSBAL) and a nonresident element (SYSSIP). The instructions in the various elements gather pertinent statistics based upon the event occurring at the point in the element. The additional Executive element SYSBAL consists of a D-bank data area and some online data reduction routines associated with certain event occurrences (i.e., subroutines that are jumped to when the related event occurs). In addition, SYSSIP provides the interface between the user, the operator and the system.

The SIP does affect the operation of the Executive. Because of this, SIP has an associated system generation philosophy. The SIP can be incorporated into the system at five different levels based upon the proper specification of system generation parameters. SIP Level 1 produces low-overhead statistics (mainly incremental counters) and some slightly higher-overhead I/O summary information. The statistics at this level are of primary interest to the user in providing the processor utilization statistics and I/O statistics necessary for optimizing the peripheral and mass storage configuration.

SIP Level 2 statistics reflect distributions of storage utilization and of program sizes in addition to the statistics associated with SIP Level 1. The effect on system operation in collecting the storage utilization statistics is considerably more than just the effect due to SIP Level 1. This is a result of the instruction execution overhead in calculating the storage utilization statistics and the additional core required for the tables to hold these statistics.

Some of the SIP Level 3 statistics are intended for the use of people working in the Executive development/modifications areas. Others are of value to users in optimizing hardware and software configurations and adjusting work loads for better system performance. In general, it is recommended that SIP Level 3 be configured along with levels 1 and 2 for initial evaluation of system activity from the user viewpoint.

SIP Level 4 is a more detailed level of SIP data. The associated statistics pertain to several very specific areas of the Executive. These statistics would be of interest only to the system programmers working in the associated areas.

SIP Level 5 is of the same type as 4, but the main storage and instruction execution requirements far out-weigh the importance of the statistics, so they are separated.

5.1.2. SIP Functional Description

5.1.2.1. SIP Level 1

Interrupts

The interrupt statistics describe the frequency of occurrence of the various types of interrupts received by the indicated processor. The interrupt statistics are accumulated by inserting incremental counters in the interrupt handling areas in the appropriate EXEC elements IH, IHESI, EIH, and TIME.

Idle Time

Instead of dropping into the standard Executive idle loop, the SIP systems contain idle activities. These are switch-listed activities operating at type and levels lower than any possible user activity. There are as many idle activities as there are processors. The lowest priority idle activity is called the system idle activity. This activity will be executed only when all processors are idle; thus the term system idle. The other activities are used in the computation of idle time per processor.

A processor spends a certain amount of its idle time in the system idle activity and the rest in one of the other idle activities.

When a processor is interrupted out of a non-system idle activity and a processor was executing the system idle activity, the latter processor must be informed to give up control of the system idle activity and assume control of the interrupted activity. The number of times this happens is given by the abnormal entries to system idle count.

Each loop of the system idle activity examines a device control (unit status) table. Each pass through the examination of all the device tables is referred to as one examination cycle. Using the number of examination cycles as a base for comparison, counts are kept of the number of times one or more devices are found busy and some mass storage device is found busy in each examination cycle. These counts are used to calculate system I/O deadlock time (all processors idle waiting for I/O),system idle time (all processors idle and no I/O in progress), and simultaneous processor idle time (all processors idle with or without I/O in progress = system idle time + system I/O deadlock time). In addition, percentages are calculated, for each I/O device, of the percent of simultaneous processor idle time and of I/O deadlock time that the device was active.

System Activity

System Activity data include the following:

1. Average runs open for batch and demand.
2. Average runs in core for batch and demand.
3. Average runs in backlog.
4. Total runs opened during the collection.
5. System sharing percentages.

SUPs

CPU, I/O, and CC/ER SUP charges are accumulated for TIP and non-TIP program types.

I/O

Data on I/O activity is collected by IOAU and channel module, by logical channel and control unit and by device. Data collected by channel module includes transfer counts and number of words transferred. Queuing information is collected on logical channels and control units. Data collected per device (unit) includes input and output request counts, words transferred in and out, device active times, request existence times, and unit queue lengths. In addition, a system-wide request count and average I/O service time are provided.

ERs

ER counts are collected for total ERs, individual Exec ERs, individual TIP ERs and individual user-defined ERs.

EXPOOL

Level 1 EXPOOL data provides the amount of EXPOOL currently in use and the amount currently allocated at the time data is written to the SIP output file.

I/O Load Monitor

The I/O Load Monitor provides a software means of regulating the I/O load to control data overruns. Because of the burst-lull nature of I/O, some data overruns as evidenced by late acknowledges should be allowed in order to maintain maximum efficiency. A maximum load level value (MAXLL) that is too low may choke the system when it is not needed while a maximum load level that is too high may bog down the sytem with excessive execution of overrun recovery code and lost time waiting for the device to return to its proper positioning for the next data transfer. The MAXLL is stored in the table LMDATA in the element IOTABL.

SIP provides two pieces of data to help the site analyze the functioning of the Load Monitor. If necessary, proper adjustment of the Load Monitor parameters can then be made. The data items collected are:

1. SBLMLL

   SYSBAL Load Monitor Load Level contains the current Load Monitor Load Level (LMLL) and is updated whenever LMLL is changed. This value is useful in determining the pattern of I/O experienced at a particular site. If the value is always much lower than MAXLL, then lowering MAXLL should be considered if data overruns are frequent.

2. SBLMQ

   SYSBAL Load Monitor Queue counter counts the number of times a control unit or MSA is queued to the Load Monitor Queue. The higher this number, the more the system is demanding the Load Monitor regulating effect. If this number is very low, then either the system load is too low to justify the use of the Load Monitor or the MAXLL is too high to initiate Load Monitoring when needed.

Counts of data overruns (late acknowledges) can be obtained from type 6 log entries (I/O error).

Depending on the above data, the maximum allowable load level (LMMAX) may have to be adjusted in the load monitor data table (LMDATA). This can be done in two ways:

1. The cell LMMAX in IOTABL can be changed to the desired level by replacing the entry in the LMDATA table and reassembling that element, IOTABL, or

2. The cell LMMAX can be changed to the desired level at 'ENTER DATE' while booting or recovering with the following C-keyin:

   C *address value*

   where *address* is the address of LMMAX, and *value* is a word with LMMAX in H1 and zero in H2.

   For example: C 64524 001234000000

   This keyin would change address 064524 to a LMMAX of 01234.

   Counts of data overruns (late acknowledges) can be obtained from type 6 log entries (I/O errors).

Communications Activity

The maximum number of line terminal groups active during a SIP collection are provided as well as active line times and number of characters transferred across each line.

5.1.2.2. SIP Level 2

Storage Utilization

SIP Level 2 presents the distribution of storage utilization as seen from the bank control packets. Each time the contents of storage changes, the dynamic allocator updates the bank control packets to reflect this change. At this point, the time-weighted statistics on storage usage are accumulated by scanning the bank control packet chain. The distribution is divided into as many as 32 storage size intervals. Information given for each interval is:

1. Tally

   Count of the number of times that a storage allocation change resulted in the amount of storage-in-use lying within the interval whose upper limit is specified.

2. Time

   The amount of time that the storage-in-use size was in the specified interval. Column 3 indicates the percent that this time represents relative to the total time that the monitoring was on.

The following two categories apply to the 1110 and 1100/40 Systems only:

3. Percent primary

   This represents (on the average) the percent of the interval size (in-use main storage) that is primary. Note that once the interval size exceeds the amount of primary storage, this percentage can never be larger than (total primary)/(interval size).

4. Percent of user primary

   This statistic is the percent of user primary space that is in use. It is computed by multiplying the interval size by percent primary, subtracting the size of the primary EXEC and dividing the result by the total amount of primary user space

Further statistics give a closer picture of who uses storage and for how long. The average downed storage is given (total/primary/extended for 1110 and 1100/40 Systems). Other statistics are collected for seven storage types. Three are EXEC-related: resident EXEC, EXEC segments and common banks; and four are user oriented: TIP, real time, demand and batch. For each type the following is given (where all the averages are time weighted):

1. Average total size

   The total amount of storage allocated to each storage type averaged out over the elapsed time (total/prim/ext for 1110, 1100/40 Systems).

2. Percent used of average available storage

   The percent each storage type used of available storage. Available is an average also, which is the total configured storage minus the average downed storage. Total/primary/extended are given for 1110, 1100/40 Systems.

See 5.1.2.3 and 5.1.2.4 for additional storage utilization statistics in SIP Levels 3 and 4 respectively.

Processor Activity

SIP Level 2 processor activity information is divided into activity types: EXEC type 0, EXEC type 1, EXEC type 3, ESI, real time, TIP, demand, deadline batch, and batch. The basic statistic for all of the nine activity types is the amount of time spent within each. Totals are then given for user time, EXEC time, and busy time (the sum of user and EXEC time). See 5.1.2.3 for additional data on processor activity if SIP Level 3 is configured.

Demand Response

The response time of the system to user request is the elapsed time between a user request at a demand terminal (carriage return or transmit) and the system's response to that request. The total system response to demand users is shown as an average response to all requests and as a distribution of the percentage of total requests that were serviced within a given time threshold. The responses are broken into four time categories when only the SIP Level 2 is configured, and into 10 categories when SIP Level 3 is configured.

DA Response

The report is a set of lines which correspond to various main storage queue priorities: TIP, demand levels 11-17, batch and deadline batch. When SIP Level 3 is configured, Demand-17 is divided into three lines depending upon the initial program size. These lines are approximately ordered by decreasing core priority so that the waiting times should increase towards the bottom of the page. The demand programs are ordered using an index which is a function of their size and 'compute-boundness'.

The report gives the average time a user main storage request waited in the DA request queue before being honored. This time is a significant part of the waiting time at a demand terminal and is correlated with the demand response time introduced above. Note that this correlation is biased when one of the following happens:

1. A demand program is swapped out when executing a command: it will request main storage more than once; hence several DA waiting times will correspond to only one response time for the demand user.

2. A demand program sends a command when the processor is already in main storage. No main storage request will be made; hence nothing will appear in the DA response statistics. This is

more likely to happen on a lightly loaded system. The following columns give the number of core requests for a given level. A breakdown of this number is given to characterize the distribution of the waiting times.

The DA response for each program type is also given as a distribution of the percentage of the total requests for each type within a given threshold. TIP and demand DA response times are broken into four categories when only SIP Level 2 is configured and into ten categories when SIP Level 3 is configured. There are four time categories for batch program types.

5.1.2.3. SIP Level 3

The statistics of this level are used in providing activity profiles of some critical EXEC areas such as EXPOOL, the dynamic allocator and the function loader. These profiles generally reflect system activity which is not directly controllable at the user level.

EXPOOL

Level 3 EXPOOL data includes counts of EXPOOL buffer requests and releases, by size and priority (L, M, H, I), and by PS, ES if primary EXPOOL is configured. Recursive (iterated) requests and releases are counted by size. Successful expansions and contractions, and the number of buffers acquired from and released back to the DA are also counted.

Non-Resident EXEC Functions

A description of the EXEC function segment activity is included in SIP level 3.

1. Function name.

2. Number of requests.

3. Number of loads of the function required from mass storage.

4. The average number of requests per load – this is computed by the data reduction program.

Storage Utilization

SIP Level 3 provides storage utilization information in greater detail than that described in 5.1.2.2 for SIP Level 2. In addition to the level 2 statistics, level 3 contains the following data:

1. For the seven storage types (resident EXEC, EXEC segments, common banks, TIP, real time, demand and batch), the percent of time that the type existed in main storage.

2. Average resident program size. For the three EXEC storage types, this is the average bank size. For the four user types this is the average in-main-storage user program size. This can be significantly less than the average size of a user activity shown in the report on CAU activity because swapping deletes some of the user's banks from storage resulting in a smaller in-main-storage average size. Total/Primary/Extended sizes are given for 1110 and 1100/40 Systems.

Processor Activity

Level 3 of SIP produces additional information on processor activity beyond that described in 5.1.2.2 for SIP Level 2. At this level, the total EXEC time will include EXEC ESI completion time, any EXEC real time (such as the SIP output routine), and any other EXEC activities running at user type-and-levels.

For the user activity types, the following additional information is obtained if SIP Level 3 is configured:

1. Average size.

   This is the average user size (time weighted) while executing. This size is determined by the Storage Limits Registers (non-1100/80), or Bank Descriptor Registers (1100/80), so banks that are not pointed to will not be counted.

2. Exec type 3 time by level

   CAU time spent in Exec type 3 activities is broken down by switching level. Fourteen categories are included, 13 for levels 1 through 13, and a final category for all other levels.

Dynamic Allocator

Detailed Dynamic Allocator (DA) statistics are intended to provide a measure of the extent of core fragmentation, the nature of the tasks which fail to be loaded and the frequency of execution of critical segments of the dynamic allocator. The following statistics deal with the task storage requirements:

1. Successful loads

   The total number of task loads (two or more banks/task) during the monitoring interval.

2. Successful loads with common banks

   The number of task loads that were completed where one or more of the banks required were common banks.

3. Times UCIMPR found set to start DA over

   UCIMPR is a flag which, when set, tells the dynamic allocator to reinitiate the scan from the start of the main storage queue chain (i.e., the main storage picture has changed requiring a rescan of the chain).

Statistics are collected on task failures for each type of main storage preference (require primary, require extended, prefer primary, prefer extended and no preference). For each type of preference, counts are kept of the number of banks that failed to be loaded and the average bank size. These statistics are separated for batch and non-batch tasks.

The next set of statistics provides a measure of main storage fragmentation. The statistics are separated into those associated with tasks which fail to be loaded because available main storage was fragmented and statistics regarding those tasks which fail due to lack of available main storage. The statistics, separated for batch and non-batch tasks are:

1. Number of tasks which fail.

2. Average size of those failing.

3. Average available main storage when failures occur.

Another set of statistics reflect the frequency of execution of critical segments of the dynamic allocator.

1. Total DA dacts

   The total number of DA deactivations.

2. Non-funct dacts

   The number of DA deactivations associated with the loading of a user program.

3. Number of times queue exists

   The number of times that a main storage queue exists (based on the number of DA deactivations).

4. Total number tasks in queue

   The total number of tasks examined in the main storage queue.

5. Average number tasks per queue

   The average number of tasks per main storage queue examination.

6. Total marked fails

   The total number of tasks examined in the main storage queue and found marked as failed.

7. Average fails per queue

   The average number of "marked-failed" tasks found per main storage queue examination.

8. Average size per task

   The average size of all the "marked-failed" tasks.

9. Average bank size

   The average bank size of the banks of the "marked-failed" tasks.

The dynamic allocator phases in which the successful task loads are made represent varying degrees of complexity in the main storage search algorithm.

Swapping

Statistics which describe swapping activity are provided. The percent of the total number of loaded tasks (unique tasks) that were suspended at least once is calculated as is the total task suspends and the total number of seconds spent by all tasks in a suspended state. (A suspended state does not necessarily require a swap but usually does.) Several additional quantities are calculated.

1. The average total number of seconds any specific task (which was suspended) spends in a suspended state (i.e., total average suspend time per unique task).

2. The average number of seconds spent in a suspended state by any task per suspend operation.

3. The average suspend time per task (including tasks which were never suspended).

4. The average number of unique task suspends per minute (ignoring multiple suspends of the same task).

5. The average number of suspends per minute (a task may be swapped more than once).

A count of true swaps to the SWAP$FILE is kept. These are in no way related to tasks. The average number of I/O's needed and the average amount of main storage per swap are kept.

DA Response

At SIP Level 3 and above, counts of initial main storage quanta exceeded are provided, describing the number of times a program loaded in main storage exceeded its main storage quantum. This number should be compared to the total number of main storage requests at the same level to determine whether the initial main storage quantum size should be modiifed.

EXEC Area Counters

The final statistics in SIP level 3 are selected EXEC area counters and are described below.

1. CCRS

   The number of times the routine chain main storage request is entered.

2. RCRS

   The number of times the routine remove main storage request is entered.

3. IILODS

   The number of task loads where every bank loaded is an initial load.

4. INLODS

   The number of loads from program files.

5. RELODS

   The number of task reloads from the SWAP$FILE.

6. REINTS

   Task loads consisting of initially loaded banks and reloaded banks.

7. LOADS

   The sum of all types of loads.

8. INBNKS

   The number of banks loaded for all initial loads.

9. RLBNKS

   The number of banks loaded for all reloads.

10. DAGIO

    The number of DA multi-bank scatter/gather I/O requests to SWAP$FILE.

11. DAIO

    The number of DA SWAP$FILE I/O requests.

12. SWAPFG

    The number of DAPA swaps due to periodic readjustment.

13. DSPI

    The number of times the dispatcher is entered at DISPI to scan the interrupt queues.

14. NXTTIM

    The number of times the dispatcher begins the scanning of the SWL queues.

5.1.2.4. SIP Level 4

SIP level 4 provides data on storage utilization by user types broken down into several size ranges and counts relating to the internal operation of EXPOOL.

Storage Utilization by User Types

For a set of user program size ranges, SIP provides the percentage of time programs existed in the size range and average percentage resident and percentage primary for each size range.

EXPOOL

EXPOOL counts include requests queued, type changes, abnormal returns, additional expansion and contraction activity, number of times EXPOOL tight flag is set, and data on the EXPOOL monitor activity.

5.1.2.5. SIP Level 5

This level of SIP is not implemented.

5.1.2.6. I/O Trace

There is an optional mode of the SIP which can be run independently or in conjunction with other SIP data collection. At least one level of SIP must be configured to allow configuration of the I/O trace. This option is referred to as the I/O trace mode. An output tape file independent of the SIP output data file is required. This trace gathers the following information for each I/O operation to drum, disk, FASTRAND drum or tape:

1. Run-id

   The original run-id of the user that requested the IO. This may also be an Executive function or the run-id "EXEC 8" if the request was performed using the EXEC PCT.

2. Filename

   The name of the file accessed.

3. Entry type

   The type of I/O request (IO$, IOW$, IOI$, etc.).

4. Program name

   If the program was loaded from LIB$, three characters of program name (ASM, FOR, etc.) are given.

5. I/O Request Time

   Time stamp of user request.

6. Device Active Starting Time Stamp

   Time stamp of issuing I/O command to device.

7. Device Active Time

   Elapsed time from I/O command to interrupt.

8. I/O Device Index

9. Number of words transferred

10. Software (user) function

11. F rel addr

    The file relative address (sector address unless the file is a word-addressable file) in octal.

12. D addr

    The absolute drum address. (For disk subsystems, the cylinder, head and record number).

13. Segment

    For those absolute reads of an EXEC segment (ABR$ by FNCCNT), the segment name is provided.

14. LIB$ Program Name

    The PAR processor provides data reduction capabilities for producing reports of the I/O trace data.

5.1.3. SIP System Influence

As previously mentioned, the various SIP Levels have different effects on the Operating System. These effects are due to the number of additional instructions executed and the additional main storage required for these instructions and the associated data areas.

The additional EXEC code created by SIP is in approximately 40 Executive elements and in the SIP resident element SYSBAL and the SIP non-resident element SYSSIP.

The approximate main storage requirements for 5 levels of SIP and the I/O trace in a 2x2 1100/80 System with a relatively large peripheral complement is (in decimal):

Resident I-bank	2200 words
Non-resident I-bank	3200 words
Assembled D--bank	400 words
Acquired D-bank	2700 words

Assembled and acquired D-banks are floated in the addressing space, and thus do not contribute to the collected size of the EXEC.

1108/1106 Systems would require less D-bank space due to smaller peripheral configurations. 1110 and 1100/40 Systems in general will need more I- and D-bank space to accommodate provisions for layered storage.

Main storage requirements for the SIP data are greatly reduced over previous EXEC releases, especially in the areas relating to storage size and I/O configuration.

EXPOOL space for double buffering of I/O trace is configurable. The size of EXPOOL buffers to be used is specified by the configuration parameter SIPIO. Two buffers of that size will be acquired for use while the trace is active, and released when it is inactive. Each trace entry is 9 words, so each buffer will hold data for

$$\frac{\text{SIPIO} - 2}{9}$$

I/O Operations

The total SIP overhead can vary greatly depending upon the environment. As mentioned above, the more I/O bound the mix, the greater the overhead incurred by the I/O trace. As the variations in storage utilization increase, the dynamic allocator fragmentation overhead (SIP analysis) increases. Experience in using SIP indicates that the expected overhead in normal operating environments with SIP Levels 1 and 2 and the I/O trace turned on should be in the range of 3 to 6 percent per processor. It must be remembered that this overhead cannot be directly correlated with a corresponding decrease in system thruput, especially in a multi-processor environment.

5.2. COMMUNICATIONS SIMULATOR (CS1100)

5.2.1. Introduction

CS1100 is a software communications simulator for the 1100 Series. Its purpose is to provide the user with the capability of simulating a complete communications environment without requiring the personnel (terminal operators) and hardware (terminals, CTMCs, GCSs) necessary to create the actual environment.

5.2.1.1. General Description

The major components of CS1100 are:

- CLS – Communications Line Simulator
- RTS – Remote Terminal Simulator
- TCL – Traffic Control Language

CLS is the aggregate of the 1100 Series Executive changes required to achieve simulated communications line data transfer by replacing the normal I/O with storage-to-storage moves. These changes include providing for the S option on the @ASG of a communications line and the creation of the Executive element COMCHN.

RTS is a real-time user program which simulates the hardware of remote terminals and, in conjunction with TCL, the operators of remote terminals. For each type of remote terminal to be simulated RTS incorporates a Hardware Device Simulator (HDS). RTS then provides the interface between each HDS and the TCL dispatcher, which controls the execution of TCL programs. RTS runs on any SPERRY UNIVAC 1100 Series Executive level 31.0 or later. It should be noted that RTS does not require CLS. RTS may run in one machine and simulate terminals via communications lines connected to another machine or machines.

TCL is a language developed to control traffic between a simulated remote terminal and the central site. Each simulated terminal is controlled by a TCL program that describes the actions of the operator of the simulated terminal. TCL programs may read images from program files or data files, send images from a simulated terminal to the central site, and receive images from the central site. Embedded in the TCL language is a set of powerful and flexible commands that may be used to examine and change images received from the central site or read from files.

5.2.2. RTS Operating Environments

RTS runs as a real-time program to simulate terminals connected to any other machine or machines including the machine on which it is running. Figure 5-1 shows the flow of data from a remote terminal to a demand run on an 1100 Series machine. The data comes into the CTMC from the terminal. From the CTMC it is passed in to the Communications Handler (COMR), and from COMR to the Communications Control Routine (CCR) that had previously requested it by ER CMI$. The CCR strips the communications envelope from the input message and passes it to RSI (Remote Symbiont Interface) as input from a demand terminal. RSI passes the image to READ$ which, in turn, passes it to the demand run.

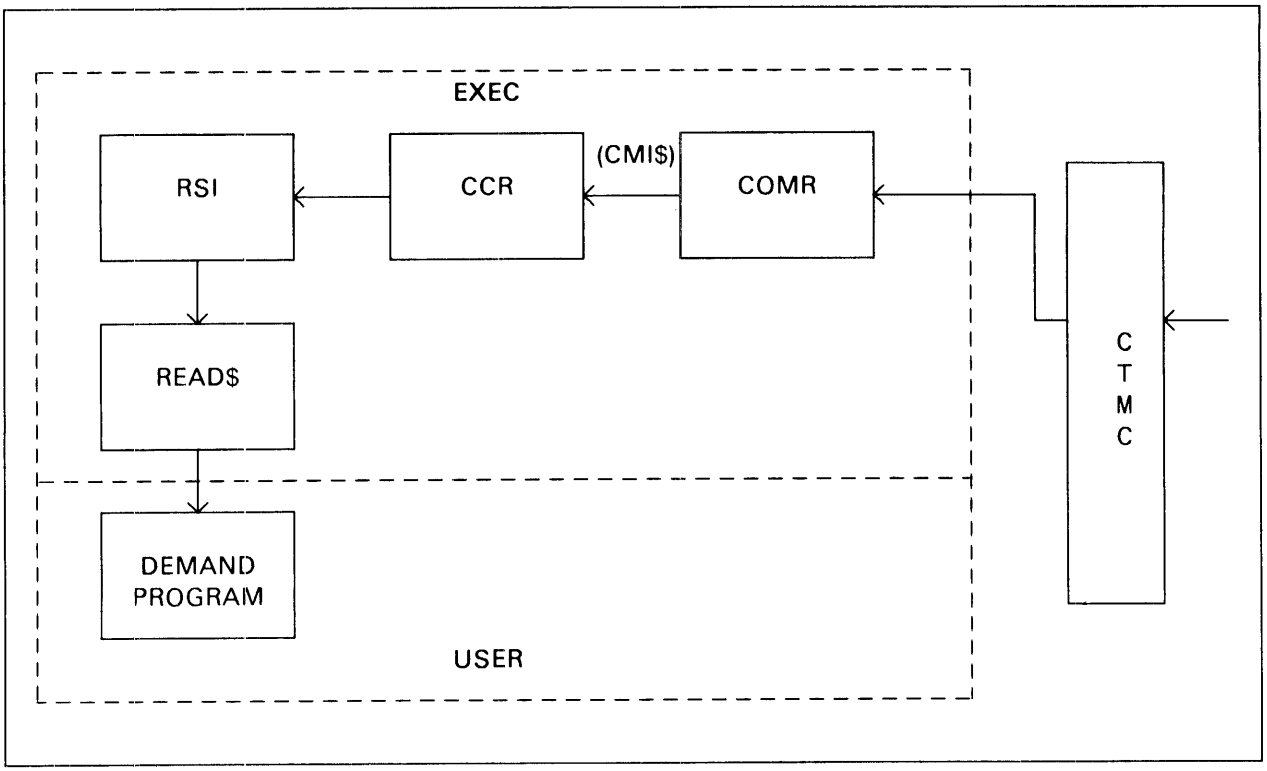


Figure 5-1. Flow of Input Data from a Terminal to a Demand Run

Figure 5-2 shows the flow of data from an RTS simulated terminal to a demand run on the same machine. In this case RTS sends the input data out to a CTMC via ER CMO$ to the communications handler (COMR). The data is turned around by two CTMs connected back-to-back in the CTMC and sent in to the Executive CCR (Communications Control Routine, i.e., device handler) that had previously requested input via ER CMI$ to the communications handler (COMR). The CCR strips the communications envelope from the input message and passes it to RSI (Remote Symbiont Interface) as input from a demand terminal. RSI passes the input image to READ$ which, in turn, passes it to the demand run.

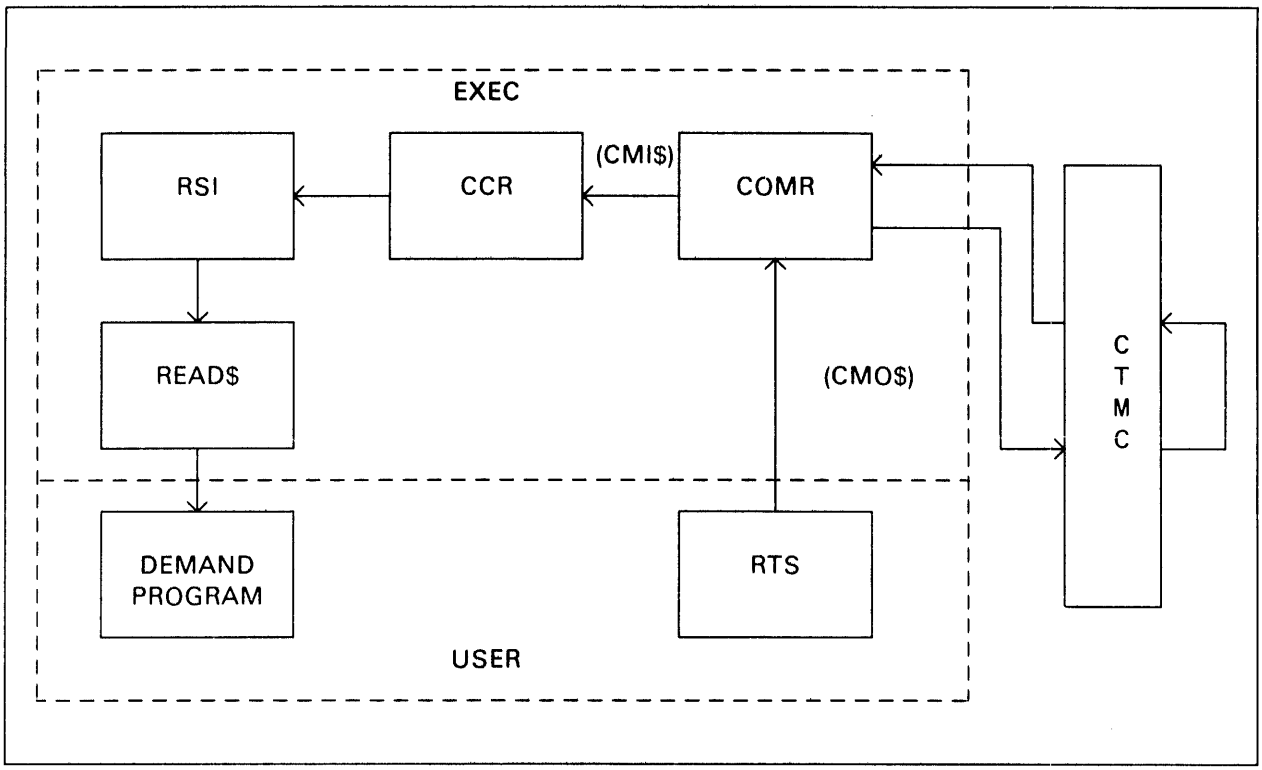


Figure 5-2. RTS Simulated Terminal to Demand Run

5.2.2.1. RTS Running With CLS

As can be seen in Figure 5-2, all data was transferred from one part of main storage to another inside the same machine. In such cases the data transfer may be accomplished entirely by software.

Figure 5-3 is an example of a communications configuration that is identical to that of Figure 5-2 except that the CTMC in Figure 5-2 has been replaced by the CLS Executive element COMCHN. COMCHN simulates a CTMC whose CTMs have all been connected back-to-back to each other in order to effect storage to storage data transfers within the same machine. Since the simulation is dynamically initiated (by the S option on the communications @ASG card) on a line by line basis, it is possible to run both real and simulated communications lines on the same CTMC at the same time.

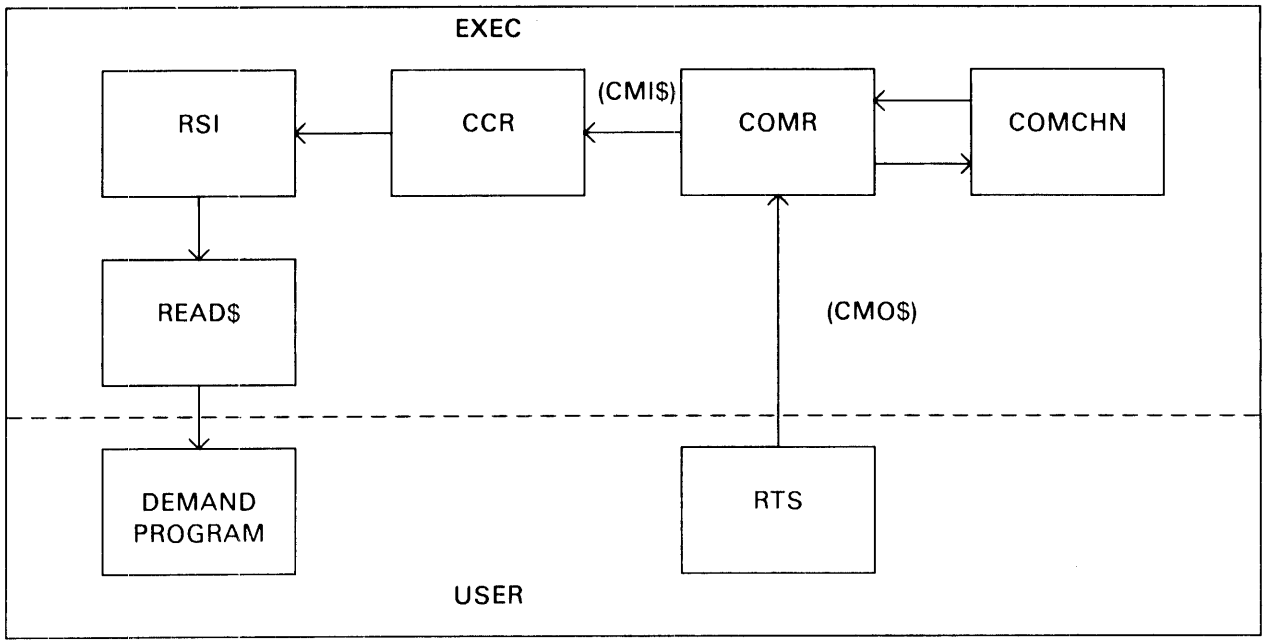


Figure 5-3. RTS Simulated Terminal to Demand Run with EXEC Element COMCHN

RTS may also send data (simulated terminal input) to another real-time program. Figures 5-4 and 5-5 show the flow of data from RTS to the real-time program. Note that in this case there is no data flow from the communications handler to the Executive CCR.

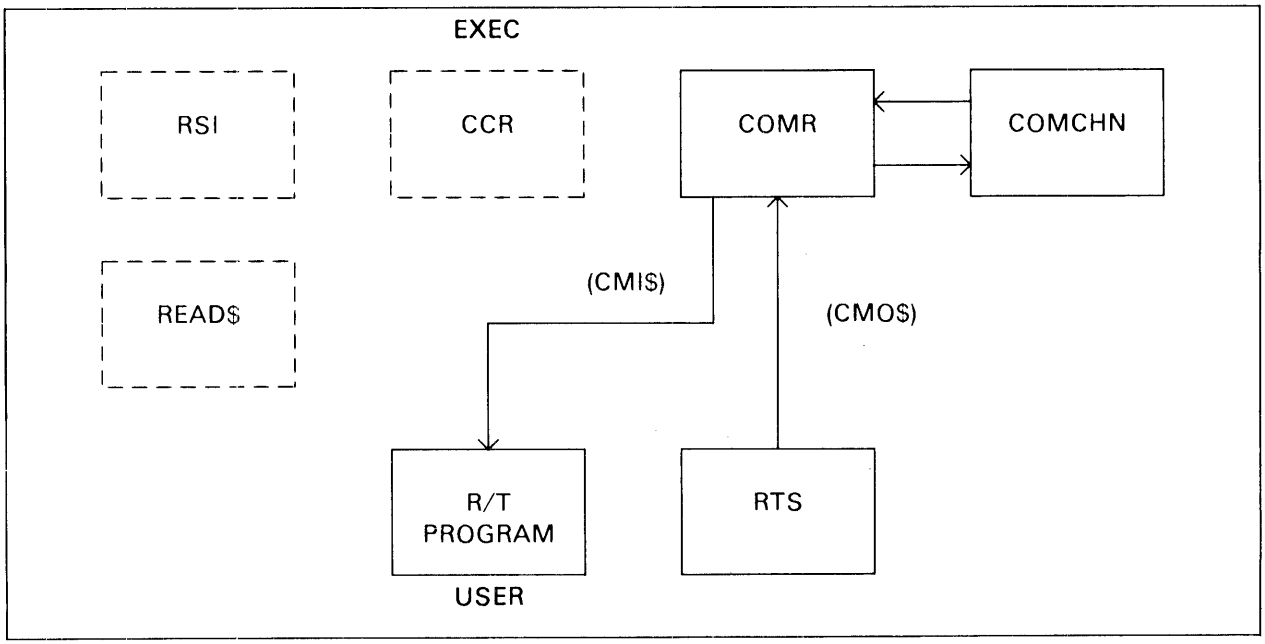


Figure 5-4. Data Flow from RTS to Real-Time Program

In Figure 5–5, the real-time program to which RTS is passing data is a user CCR using the General CCR (RSI$) interface.

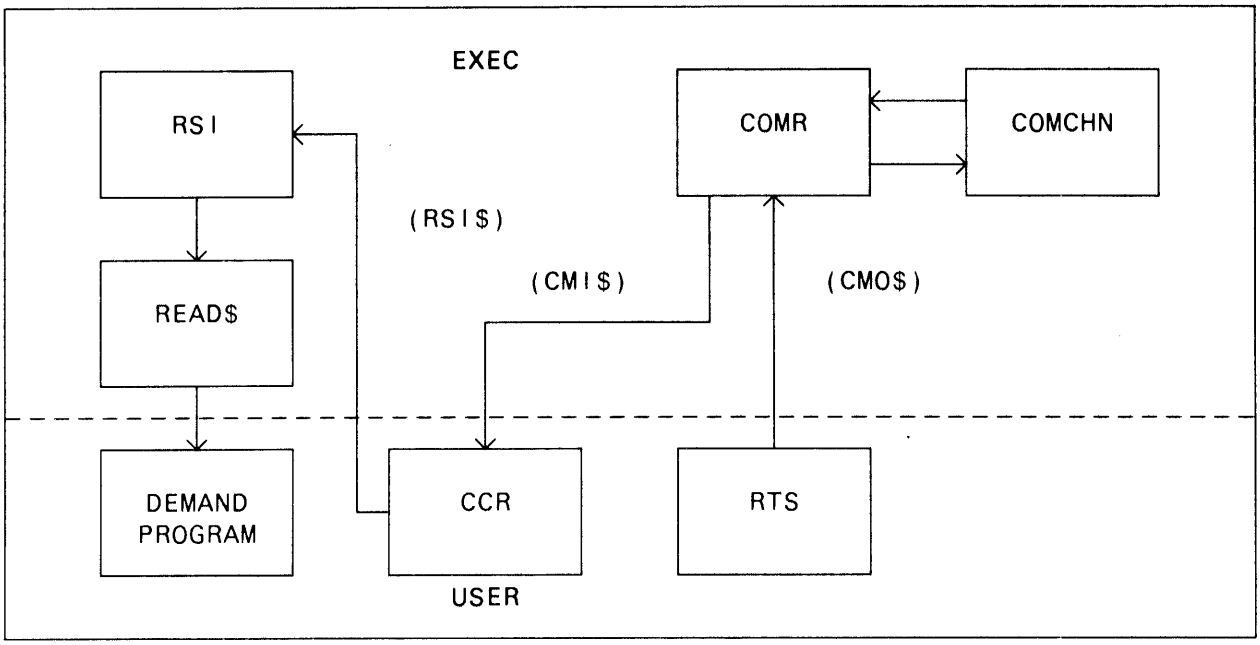


Figure 5–5. Data Flow from RTS to Real-Time Program Using RSI$

Using the RSI$ HDS (which is invoked by the *RSI command), simulated terminal input may be sent to a demand run in the same machine in which RTS is running without going through the communications handler at all. The data is passed from RTS directly to the Remote Symbiont Interface (RSI). If the RSI$ HDS is the only HDS being used then RTS need not run as a real-time program but normally does so for the following reasons:

1. It is theoretically possible to cause a system hang.

2. If the demand load is sufficiently heavy, RTS may be swapped out of storage for more than 10 minutes in which case RSI would terminate all of the simulated terminals for lack of activity.

Figure 5–6 illustrates the flow of data from RTS to a simulated demand run when the RSI$ HDS is used. Note that in this case the communications handler is bypassed entirely as the data goes from RTS directly to the Remote Symbiont Interface (RSI).

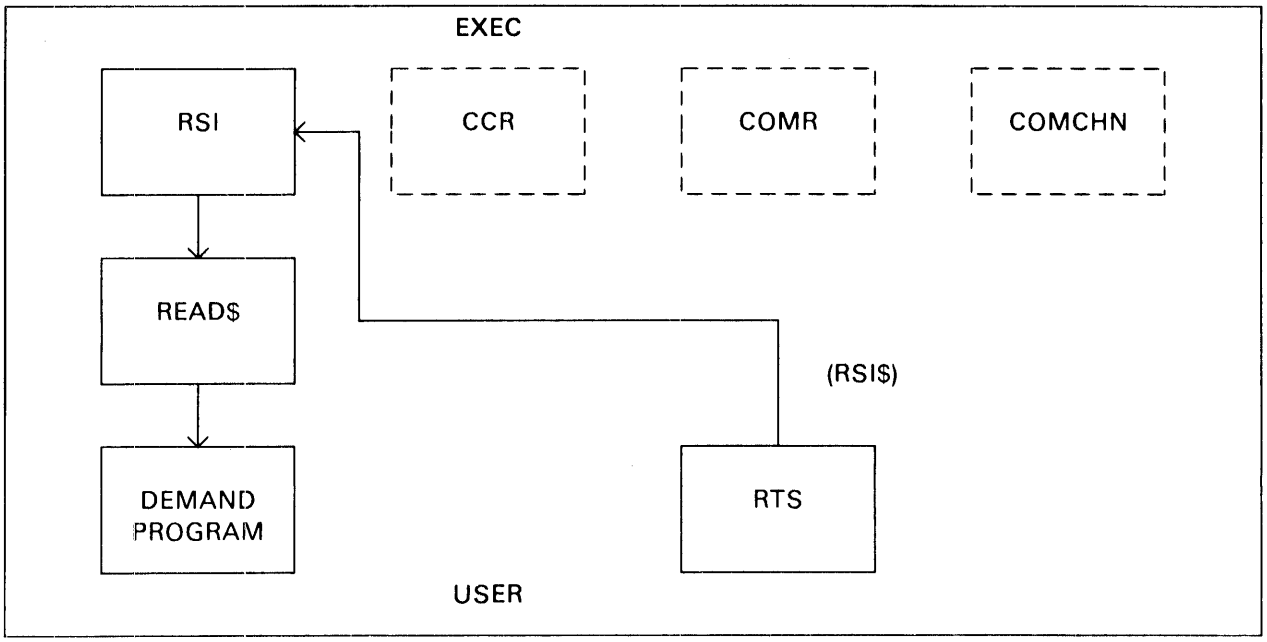


Figure 5-6. Data Flow from RTS to Simulated Demand Run Using RSI$ HDS

5.2.2.2. RTS Running Without CLS

While it is often convenient to have RTS use the Communications Line Simulator interface (CLS), it is by no means a requirement. This subsection illustrates some of the many configurations in which RTS may run. None of the configurations discussed require the use of CLS.

In the configuration illustrated in Figure 5-2, although RTS is running in the same machine as the simulated demand run, CLS is not used. Instead the data is sent out to a CTMC that has had two of its CTMs connected to each other in order to pass the data back into the same machine. While the CTMC in this configuration can be simulated by CLS, a real CTMC may be used if desired.

The configuration illustrated in Figure 5-7 shows RTS running in one machine simulating TIP terminals connected to another machine by connecting communications lines from the machine in which RTS is running to the machine in which CMS is running.

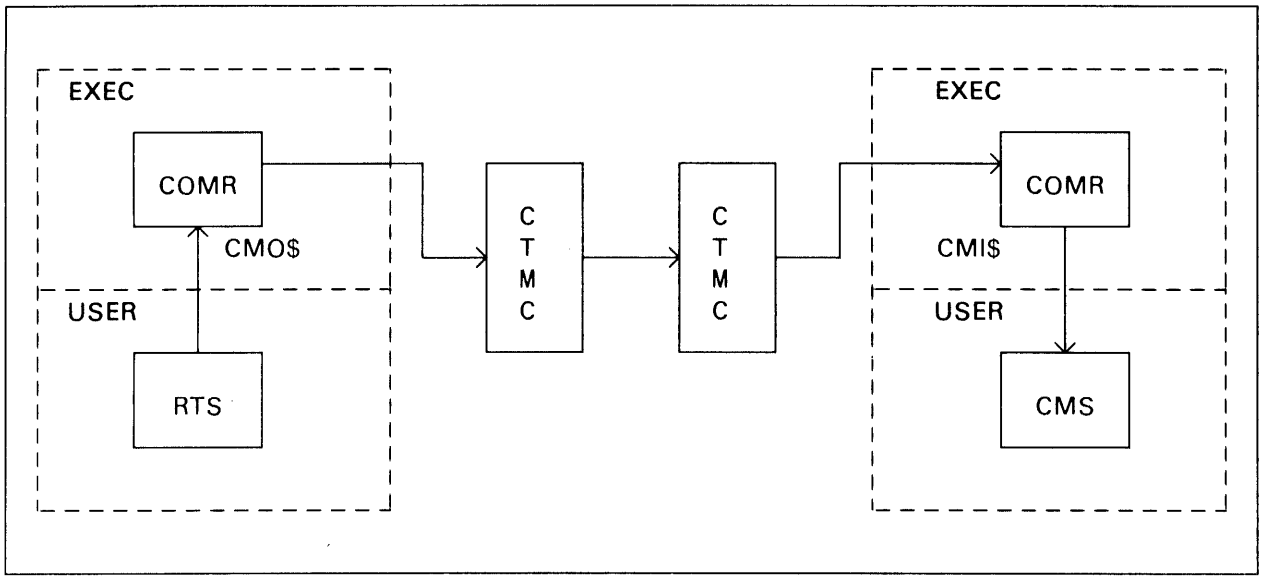


Figure 5-7. RTS in Dual Machine Environment with CTMC

The configuration shown in Figure 5-8 is identical to that in Figure 5-7 except that the CTMC connected to the machine in which CMS is running has been replaced by a C/SP.

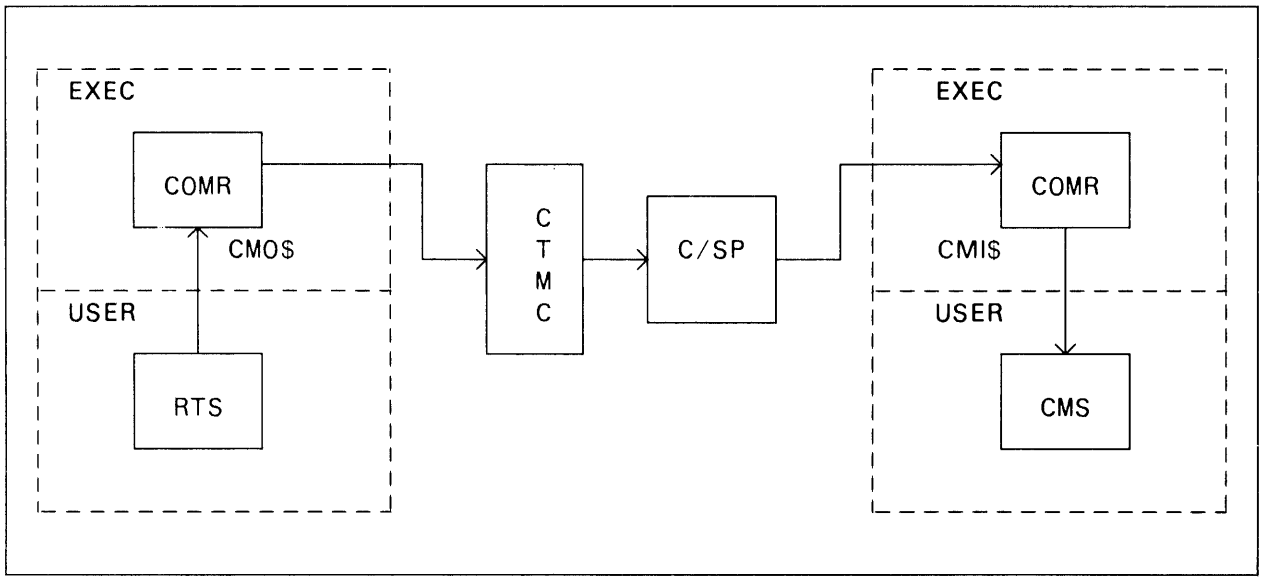


Figure 5-8. RTS in Dual Machine Environment with C/SP

5.2.3. Release Tape Format

The CS1100 release tape is an eight file tape. The release tape format is as follows:

File Number	Contents	Remarks
1	utility file	Contains skeleton (standard SGSs, etc.).
2	base level	Base level symbolics.
3	PCF	Permanent Correction File.
4	updated symbolics	Updated (current) symbolics.
5	RB file	Contains all current relocatable and absolute elements.
6	run file	Contains all elements necessary to run RTS and TCL, as well as technical documentation for RTS.
7	print file	PDP and ASM print of current symbolics.
8	TCL file	source relocatable, and absolute elements comprising the TCL compilier.

NOTE:

All files are in @COPY,G format.

5.2.3.1. The Run File (File 6)

File 6 of the CS1100 release tape contains all that is necessary to run RTS, take dumps of RTS, process the RTS log file, and compile TCL programs. It also contains some utility run streams and technical documentation for RTS. The technical documentation for RTS is contained in the @DOC format element RTSDOC. The following runstream will catalogue this file:

```
@RUN        RUNID,ACCT,PROJ
@ASG,CP     RTS,/1//512                . RTS WILL CONTAIN FILE 6
@ASG,T      CSXXX,T,YYYYY              . CSXXX IS THE RELEASE TAPE
@MOVE       CSXXX.,5
@COPY,G     CSXXX.,RTS.
@FREE       CSXXX
@FREE       RTS
@FIN
```

5.2.4. System Configuration For CLS

The Executive configuration changes required to use the Communications Line Simulator (CLS) or the general CCR Interface (RSI$) may be found in the current CS1100 Software Release Documentation.

5.2.5. RTS Execution

5.2.5.1. Processor Call

RTS must be called as a processor and not with an @XQT image. The processor call for RTS is

```
@RTS,<options>
```

5.2.5.2. Options

The options are:

C Solicit input from the onsite operator console.

N Suppress printing control statements submitted to RTS.

If the C option is specified, RTS will solicit input to the User Control Interface from the onsite operator console with a type and read of the form:

```
0-runid*RTS?
```

Control statements may then be typed in. A zero-length response to the type and read will turn off the console solicitation. If the C option is not specified on the processor call or if console solicitation has been turned off, input may be provided from the onsite operator console by interrupting the program with an unsolicited II keyin. RTS will respond with a type and read of the above form. Control statements may then be typed in. If the C option was not specified on the processor call, many of the RTS diagnostic messages will be placed in the PRINT$ file of the run in which RTS is running rather than on the onsite operator console. The C option tells RTS that the user of RTS is at the onsite operator console and wishes to be notified of all changes in the state of the simulation. The absence of the C option indicates that the user of RTS is not at the onsite operator console and that no messages are to be directed there.

5.2.5.3. Control Statements

The actions of RTS are directed by control statements provided either in the READ$ file or from a demand run or from the onsite operator console. A brief summary of control statements is provided. For more detailed information see the current CS1100 Software Release Documentation.

- *ABORT - Abort Simulation

  The *ABORT command will cause RTS to do an immediate abort.

- *ADD - Dynamic @ADD through CSF$

  See SPERRY UNIVAC 1100 Series Executive System, Volume 2, EXEC Programmer Reference UP-4144.2 (current version), for a description of @ADD. The *ADD card can be used to direct RTS to read control statements from a file or element that is catalogued on the system on which RTS is running.

- *ASG - Dynamic @ASG through CSF$

  See SPERRY UNIVAC 1100 Series Executive System, Volume 2, EXEC Programmer Reference UP-4144.2 (current version) for a description of @ASG. This control statement directs RTS to

do a dynamic @ASG of the file specified on the *ASG card. The *ASG command may be used to @ASG a file to the RTS runstream prior to @ADDing it to the runstream if the file contains elements that are to be used as Transaction Control Files (TCFs).

- *AT – Get a Count of Active Terminals

  The simulator will display the decimal number of active terminals. A terminal is considered to be active if it has been configured; e.g., an idle UNISCOPE 100 terminal that is not being polled is considered to be active.

- *BANKS – Display Size and Location of RTS Banks

  The *BANKS command will cause RTS to display the absolute main storage block address and size of all of its banks.

- *BRKPT – Dynamic @BRKPT through CSF$

  See SPERRY UNIVAC 1100 Series Executive System, Volume 2, EXEC Programmer Reference UP-4144.2 (current version) for a description of @BRKPT.

  *BRKPT is useful after a *ADD has been done to print any diagnostic messages that might have been placed in the run's PRINT$ file after the *ADD occurred. All diagnostic messages pertaining to the control statements in the @ADDed element will be placed in the PRINT$ file. If not all of the control statements appear to have been acted upon properly, a *BRKPT command can be used (in conjunction with *SYM) to send the run's PRINT$ file to an onsite printer to determine the cause of the problem.

- *CAT – Dynamic @CAT through CSF$

  See SPERRY UNIVAC 1100 Series Executive System, Volume 2, EXEC Programmer Reference UP-4144.2 (current version) for a description of @CAT.

- *CU – Display Internal Core Usage

  The *CU command will cause RTS to display a summary of internal main storage usage.

- *DCT – Simulate DCT 500 in Teletype Mode

- *DCTM – Simulate DCT 1000

- *DELAY – Set Delay Rate

  The *DELAY command is useful when control statements are coming from the READ$ file. It may not be desirable to initialize fifty or a hundred simulated remote terminals at exactly the same time. The *DELAY command establishes a delay time between the reading of control statements from the READ$ file.

- *FDB, FDD, FDS – Simulate a full duplex remote 9300 System site (NTR)

  Three types of control statements must be submitted in order to simulate a full duplex remote 9300 System (NTR) site.

- *FLAP – Instrument an RTS Activity with Flow Analysis Program (FLAP)

  The *FLAP command will cause RTS to force an internal activity specified by the user to instrument itself with FLAP.

- *FREE – Dynamic @FREE through CSF$

  See SPERRY UNIVAC 1100 Series Executive System, Volume 2, EXEC Programmer Reference UP-4144.2 (current version) for a description of @FREE.

- *LG – Dynamic Logging Control

  The *LG command has been implemented in order to dynamically control logging of information in the RTS log file.

- *MSG – Display Message on Onsite Operator Console

  The *MSG command will cause RTS to display a message on the onsite operator console.

- *QUAL – Dynamic @QUAL through CSF$

  See SPERRY UNIVAC 1100 Series Executive System, Volume 2, EXEC Programmer Reference UP-4144.2 (current version), for a description of @QUAL.

- *RSI – Simulate Remote Terminal through ER RSI$

  The *RSI command will simulate a demand terminal through the general CCR (RSI$) interface.

- *SETBP – Set Hardware Breakpoint Register

  The *SETBP command is an RTS internal debugging aid. It results in the setting of the 1110, 1100/40 or 1100/80 System's hardware breakpoint register via ER SETBP$. In addition, action to be taken upon occurrence of the breakpoint interrupt may be specified. See SPERRY UNIVAC 1100 Series Executive System, Volume 2, EXEC Programmer Reference UP-4144.2 (current version) for a description of the hardware breakpoint register and ER SETBP$.

- *START – Dynamic @START through CSF$

  See SPERRY UNIVAC 1100 hardware Series Executive System, Volume 2, EXEC Programmer Reference UP-4144.2 (current version) for a description of @START.

- *TTY – Simulate Teletype

- *USE – Dynamic @USE through CSF$

  See SPERRY UNIVAC 1100 Series Executive System, Volume 2, EXEC Programmer Reference UP-4144.2 (current version) for a description of @USE.

- *U100 – Simulate UNISCOPE 100 Display Terminal

  The *U100 command may be used to initiate the simulation of single station, multidropped, multiplexed, cascaded multiplexed, or multidropped multiplexed UNISCOPE 100 terminal networks.

- *U200 – Simulate UNISCOPE 200 Display Terminal

- *WAIT – Pause Between Control Statements

  The *WAIT command establishes a pause for a specified number of seconds before reading the next control statement. If delay time is also in effect, the *WAIT directive overrides the *DELAY directive.

- *X – Terminate a Simulated Terminal

  The *X command may be used to terminate one or more simulated terminals.

5.3. 1100 SERIES TEST PACKAGE (FIREUP)

5.3.1. Introduction

The purpose of the 1100 Series Test Package is to provide Systems Software Development with a level of confidence in the Operating System. This test package is made available to 1100 Series System users as Category 3 software in order to enable users to guard against system regressions as local code is added or a new Executive level is installed. It is also performing useful system analysis, such as comparing one Executive level to another, comparing one site to another, or a site to itself when the configuration is altered.

The quality of tests in the 1100 Series Test Package is a matter of continuing importance. As new features are incorporated into the Operating System, the testing system stands in danger of becoming obsolete unless additional tests are incorporated to test these new features. It is toward the goal of software integrity that Systems Software Development continues to strive. We recognize, however, that every configuration combination cannot be tested at one site and that the test package does fall short of 100% testing. For this reason, Systems Software Development solicits suitable new tests from sites who will take the time to format them.

5.3.2. Functional Overview

The 1100 Series Test Package is a collection of tests and assorted test tools. To better understand the need for test formats, an overview of existing test tools is provided, followed by test package formats, operations and maintenance.

5.3.2.1. Objectives

The test tools developed for the 1100 Series Test Package were designed with three goals in mind:

- Minimum Test Format Considerations
- Minimum Print Output
- Maximum Visibility into Test Sessions

Conversion of a @RUN or sequence of operations (procedure) to 1100 Series Test Package format is made as simple as possible. A test run can be added to the test system via the skeleton provided following normal Executive update disciplines. As a minimum, three Executive control statements (@ADD) should be added to the run before insertion. Procedural tests should be formatted into a step list with expected responses and comments. Formatting details can be found in the 1100 Series Test Package release documentation.

Several techniques are described in the release documentation and discussed briefly to minimize print output and allow the site manager to see the results of a test session.

5.3.2.2. Test Tools

The 1100 Series Test Package uses S4 and S6 of the condition word to log and keep track of run status within each run. Test programs to be included in the test package may use S3 and S5 of the condition word for their own purposes, but should be careful to avoid inadvertantly destroying either S4 or S6 of the condition word through such sequences as @SETC 0100. Specific uses of the condition word by each test tool are described in the Test Package release documentation.

5.3.2.2.1. The STATUS Processor

In an effort to make system testing as automatic as possible, a program has been developed that will summarize the results of a test period or a series of test periods. Its primary input is the master log. Its function is to determine the status of each run and print summary listings of all runs that terminated in error, all runs that terminated normally, etc. A mechanism exists to automatically call STATUS at timed intervals during the test session. The call to STATUS includes source commands to direct STATUS in preparing the test session summary. The following are listed for each run, so that it may be possible to determine the results of a particular test without having to actually find the output of the test:

1. FIN time
2. total SUPs time, CPU/CAU time, I/O time, CC/ER time, voluntary wait time
3. site-id, if demand
4. time in RT mode
5. cards in, pages out, cards out
6. last reentry address if terminated in error
7. condition word
8. Executive level

STATUS was designed in conjunction with a set of subroutines and procs that reside in the test library. The function of these subroutines and procs is to enable test programs to record their status in their condition word at any point during a run.

STATUS recognizes the following conditions which may be logged by the calls to the subroutines and procs.

1. Normal Status

   A test or some part thereof has been successfully completed and no error has been detected.

2. Failure Status

   A test or some part thereof has failed.

3. Abnormal Status

   An abnormal condition has been detected whose cause cannot be determined or lies outside of the province of this particular test. For example, if the dynamic assignment of a file failed during a test of I/O, the status should be logged as abnormal rather than failure.

4. Runstream Analysis Terminated Status

   A test expects to error and expects Executive standard action to be taken on the error. STATUS will later verify that an error occurred.

5. MCT Abort Status

   A test has been aborted by the AVAIL processor (see 5.3.2.2.3) because the equipment and/or system required for the test was not available.

The subroutines and procs enable test programs to automatically log the current status of a run, place diagnostic messages in either the print file or the master log and, if conditions warrant, terminate the run. The test package has various methods for status logging to meet special needs of tests. There exist status logging methods from Assembler elements, methods from higher level languages and a method from the runstream.

5.3.2.2.2. Print Control

With STATUS providing vital information on each run executed during a test session, the print output in most cases need not be inspected. A mechanism was devised using the run's condition word and Executive control statements to determine during the run whether the output should be printed or not. This mechanism consists of three @ADD elements (INITIAL, MID, and FINAL) to be placed into the test's runstream.

INITIAL must be added immediately following the @RUN statement, MID must be added after each @XQT and FINAL must be added immediately preceeding the implied @FIN. If the status logging subroutines and procs are not used, printout will be allowed only if the test entered error mode. Thus, the print control mechanism will function even without status logging.

The @ADD elements have taken on additional duties beyond that of print control. Of particular note is that of initialization of other test tools and assignment of test package files. The contents of the @ADD elements are subject to change. Their use and contents are fully described in the release documentation.

5.3.2.2.3. The AVAIL Processor

It has been found that many tests are hardware or software dependent in that if a particular equipment type or Executive level is not available the test is not valid.

The AVAIL processor, which is part of the 1100 Series Test Package, accepts parameters from individual tests at run time, tests for desired conditions and returns control at predetermined points in the test. This program uses the Executive's Master Configuration Table (MCT) to determine availability.

Calls to AVAIL support both AND and OR conditions in the same call. In the normal case if the test fails, control drops through to the control statement following the call to result ultimately in abandoning the run. If the conditions are satisfied, control skips to the second control statement following the call for normal continuation. Other cases include the capability of setting the condition word on pass or fail and the ability to re-enter the runstream elsewhere via a label.

As with the other test tools, complete user documentation is available as part of the software release.

5.3.2.2.4. SCOMP (SDF Comparator)

In an effort to deal with tests which cannot be made self checking, the SCOMP processor was implemented for the test package to compare the print files of tests. The implementation provides automatic comparison for those tests whose base print files are available at test session time. Because the comparison occurs within the runstream of each test (as the new print file is being created), many tests or multiple copies of test may be simultaneously calling SCOMP with no conflicts. The SCOMP call has either two or three specification fields: the first two are the files compared and the third, if given, is the difference file.

Available SCOMP options are:

- Set condition word if unexpected differences are found.
- Drop deleted images.
- Strip out control images.
- Some images may be ASCII (print files only)
- Reduced compare (only compare spec1 text within spec2 text).
- Strip trailing blanks from images before comparison.
- Write SDF text of differences into spec3 (usable by SIR).
- Ignore image control word when comparing images.

Some of the objectives set for the design of SCOMP and its use with the test package are:

1. Fast and efficient image matching.
2. Produce a temporary correction file from a base element and an updated element.
3. Capability of comparing only a portion(s) of a test's print file.
4. Ease of creating and sanctioning new base elements.
5. Status logging of comparison results.
6. Minimal changes to a test's runstream.
7. Automatic comparison of tests whose base exists, but with the capability to prevent comparison.
8. Print control.

5.3.2.2.5. Common Banks

The test package has special common banks in SYS$*LIB$ at Roseville Development Center for testing purposes. These common banks have BDIs and options defined on REPROG cards in the element AACONFIG. The common bank BDIs and entry points are also defined in elements R$PDEF and CERU$ in the test package. Hence, tests may reference the common banks without other

Executive library changes. Note, however, that these common banks, COMMND, CMNREP, TSTIBK and TSTDBK are not officially released common banks and that their BDIs must be occasionally changed to prevent BDI conflicts.

5.3.2.2.6. Procedural Test Retrieval

A large part of testing an operating system is made up of performing sequences of operations and reacting to responses such as console keyins and tape and disk operations.

A program in the 1100 Series Test Package retrieves and displays specially formatted procedural tests. Each procedure includes a place for keywords for retrieval purposes. The Procedural Test Retrieval program will list test names on keywords, retrieve and list tests on test names, list a complete table of contents with or without keywords, etc.

The complete user documentation is included as part of the 1100 Series Test Package release.

5.3.2.2.7. Tape Labelling

Privileged utility programs to provide pre-labelled blank tapes or unlabelled tapes for tests are part of the 1100 Series Test Package.

5.3.2.2.8. Communications Simulator (CS1100)

The 1100 Series Communications Simulator, CS1100, has been developed at Roseville Development Center to provide for demand testing. CS1100 is not now included in the Test Package. It is maintained separately with separate documentation and releases (see 5.2).

5.3.2.2.9. Console as a Remote Terminal

A real-time communications test program is part of the 1100 Series Test Package and allows the 1100 Series operator to use the system console as a remote terminal. This program (TTY) requires the Communications Line Simulator (CLS) portion of CS1100.

5.3.3. 1100 Series Test Package Operations

The test package is catalogued as a read-only, unload inhibited file so that several runs may simultaneously access procs and subroutines from the file and so that those runs which are designed to create mass storage tight conditions will not cause the test package to be rolled out. The test package program, STATUS, is started every 30 minutes of testing to summarize and print results of the testing period. A runstream to catalogue the test package is provided as part of the release.

5.3.3.1. Test Selection

5.3.3.1.1. The Test Package Directory

The Test Package Directory provides the following information about each test:

1. Test number
2. Test name
3. Run time in SUPs
4. Pages and cards out
5. Expected termination
6. Brief description

The directory also lists start elements, tools, and supporting elements. The run DIRECT prints the directory and also produces a cross listing of tests.

5.3.3.1.2. The Start Elements

The test package contains runs which start groups of tests. These runs are the start elements. Sequential start elements exist to allow tests within a range of numbers to be started. More important are the start elements which start tests in a particular Executive area. Start elements which start a half hour of testing in various Executive areas have been devised. These half hour tests are undergoing changes and are documented in the directory.

5.3.3.1.3. Cross-Reference of Tests

When included in the test package, tests are given a unique test number. This number is used both as the element name and the runid. A cross reference program provides the capability of finding a test by its original test name.

5.3.3.1.4. The Start Keyin

Once the test package has been catalogued, tests and start elements can be started by the keyin:

ST TEST*TEST.*xxx*,SETC,,account number

where:

xxx Test name or start element name.

SETC A flag field which, if set to 1, forces printout regardless of type of termination.

Procedural tests are printed via the Procedural Test Retrieval program.

5.3.3.2. Test Package Maintenance

The 1100 Series Test Package consists of:

- Test File

  All current 1100 Series tests.

- Document File

  All test package documents.

- Base File

  The base print output for tests whose print files are examined by SCOMP.

The test package is maintained via SSG much the same as Executive maintenance. A skeleton is provided as part of the 1100 Series Test Package release. To use the skeleton to maintain the test package, one simply builds the update tapes using the SGSs and runstream as documented.

5.3.4. Test Package Format and Conventions

Test programs to be included in the 1100 Series Test Package should conform to guidelines established to keep the test package in a form that is easily used and maintained. The guidelines are as follows:

1. The programs should be self-checking. If any errors occur, the incorrect values should be printed out along with the expected values and any other pertinent information.

2. A description of the run should be included on comment cards in a separate element within the run. It should define what is tested and what results are expected. Also, any documentation necessary to operate the run should be included.

   The documentation should include the area covered by the test and the objective of the test, including the details of test construction. All requirements for unusual or specific facilities should be listed. Any unusual operational procedures should also be noted.

3. Where possible, each test should be dependent only upon those functions to be tested. Where this is not possible, the number of external dependencies should be minimized.

4. The tests should be as self-contained and as 'automatic' as possible. They should not reference any previously prepared tapes or other pre-stored data. This data should instead be included in the test by using @ELT. Execution via conversational terminals should be avoided when the same functions can be achieved in a batch environment.

5. Environment conditioning (special setup of hardware configurations) and external intervention (operator action) should be minimized and should be isolated in as few tests as possible.

6. Serial dependencies between or within tests are to be avoided unless it is the dependency itself which is being tested.

7. All code should be thoroughly commented.

8. Runs should clean up their refuse – e.g., files temporarily catalogued for a particular test run should be decatalogued by that run before it terminates.

Five factors used in rating a run as a 'valuable' test are:

- Uniqueness of Function

  Does this run exercise some feature or features of 1100 Series Software more rigorously than any other run in the package?

- Follow Through of Operations

  Does the run execute most of the code it compiles? Does it produce easily ascertainable evidence in printout summary, or in some other manner, that all steps in the run were correctly taken?

- Efficiency

  Does the run, in a relatively tight time frame and printed page count, perform large numbers of varied actions, as opposed to heavily repetitive executions that do a small number of actions many times, without imposing a useful cumulative demand on the system?

- Scope

  Is the run large enough to stand on its own, or could its significant portions be grafted without undue work into a larger, more general purpose run?

- Ease of Run Submission

  Except for runs specifically designed to thoroughly and strenuously check out EXEC tape handling, keyin reaction, card punching, starting of sequence runs, and other behavior that may require special operator cognizance, the typical runstream is ideally stripped down so it can be started without prior knowledge of its contents. Often small changes in coding can effectively simulate external inputs. If a run must read its data in from an independent tape, a setup routine can be prefixed to the run which gets the data from the runstream and puts it out on a scratch tape for initialization. If the run is device independent, a temporarily assigned mass storage file might serve logically the same purpose as a tape or removable disk file.

5.4. 1100 SIMULATOR (FLIT)

5.4.1. Introduction

The 1100 Simulator (FLIT) provides online debugging capabilities that emulate the actual "hands on" use of the 1100 Series hardware. Through the 1100 Simulator, the user has an interface that is directly analogous to being on the actual system.

This is accomplished by building, within FLIT, a virtual processor environment which matches the machine type and peripheral configuration of the machine being simulated. This virtual processor is independent of the host processor running FLIT. The ability to divorce the virtual processor configuration from the host processor configuration allows software debug of operating systems or device handlers without having the physical hardware configured onsite. The boot tape used as input to FLIT must, however, be configured to match the configuration of the virtual processor.

Through the simulation of device error generation, the recovery software related to each peripheral can be debugged.

One of the most important attributes of the 1100 Simulator is the savings in CPU time for debugging. Without this capability, the allocation of an entire 1100 System is required to checkout new software. With the 1100 Simulator, however, multiple Series boots and debugging sessions can be done simultaneously and in an environment identical to a bootstrap operation that would occur on the actual computer system.

5.4.2. The Design Criteria of FLIT

In the design of the 1100 Simulator, emphasis was put on developing a simulation capability that was relatively fast and that was simple and easy to use. Modularity was another important design consideration, so that future simulation development and modifications could be accomplished with a minimum of effort. Guidelines in the implementation were that no module should perform a function which is logically the function of another module, and adhere to a rigid interface discipline between modules.

By achieving these objectives, an exceptionally versatile and powerful debugging tool was produced. The 1100 Simulator, FLIT Fault Location by Interpretive Testing), is a generalized SPERRY UNIVAC 1100 Series user program which provides an interpretive execution and interactive debugging capability for both user programs and Executive routines written for the 1100 Series computer. The simulation of the total addressing environment and instruction repertoire of this computer is performed.

FLIT executes as a user program under the 1100 Series Operating System. It is written in assembly language and its I-bank is structured as a common bank. It relies heavily on the multi-activity and demand mode features of the Executive. The multi-activity feature allows FLIT to define multiple virtual processors for use in the simulation process; and the demand feature allows for interactive communication with FLIT while it is executing. Most importantly, FLIT can operate under any 1100 System; i.e., 1106, 1108, 1100/10, 1100/20, 1110, 1100/40, 1100/80, etc., so that no special software or hardware is necessary to run it.

The simulation process is accomplished via a 2-activity program. One activity, called the operator activity, provides the FLIT user interface for defining the machine configuration and controlling the actual simulation process. The operator activity is the master activity within the simulator and performs all functions that would be logically associated with the machine operator. The second activity, referred to as the machine activity, contains the components necessary to simulate the machine environment and is a slave to the operator activity. These activities run asynchronously when executing in demand 80 System; i.e., 1106, 1108, 1100/10, 1100/20, 1110, 1100/40, 1100/80, etc. allowing constant monitoring and complete control of the simulation process by the FLIT operator. (The FLIT operator is the user.)

In batch mode, synchronization of the two activities is provided by the ability of the operator activity to deactivate itself until one of a number of particular events occurs to halt the machine activity. At that time, the operator activity is re-activated and determines whether processing should continue or the program terminated.

5.4.3. Simulation Modes

FLIT is capable of operating in three modes: program mode, system mode and PMD mode.

5.4.3.1. Program Mode

Program mode allows loading of an 1100 Series EXEC absolute which then executes as if it were under control of a limited Executive. In the program mode, most Executive Requests are passed on to the host 1100 Series Operating System for processing. Those that are not processed cause suspension of the machine activity, allowing a decision by the FLIT operator whether to continue execution or to terminate the program.

5.4.3.2. System Mode

In the system mode, storage modules, I/O devices and the system console can be configured to match the desired execution environment. An "initial load" command is provided to load an EXEC boot tape into the simulated storage. Once the first block of the boot tape is loaded, the system being simulated handles all interrupts defined for the machine configuration.

In system mode, as well as program mode, it is impossible for the executing routine to determine whether it is executing on actual hardware or under FLIT.

5.4.3.3. PMD Mode

In PMD mode it is possible both to take dumps and to perform on line debugging of a user program. A special load directive is available which permits the dump of an absolute program, which was saved in the DIAG$ file, to be loaded and initialized to the state that the program was in when it was originally dumped.

5.4.4. Internal Logic

The internal logic of FLIT is controlled by the machine activity and is broken into a number of discrete modules. Each module is self-contained and represents a portion of the machine being simulated. The six simulation modules are:

- Storage
- Processor
- Input/Output
- System Clocks
- Device Control
- System Console

With this type of structuring, future extensions to the FLIT processor to accomodate new computers or peripherals can be done with relative ease.

5.4.4.1. Storage Simulation

Storage simulation is via a buffered storage system with backing storage being a mass storage file. The mass storage file is large enough to accomodate the full 24-bit addressing range of the 1100 Systems; however the buffer is much smaller and contains an ever-changing subset of the information on the mass storage file.

The buffer is broken into smaller sets called pages. Each page contains *n* words and there are *m* pages in the buffer. Both the page size and number of pages are configurable within FLIT. Currently, settings of 28 words per page and 500 pages are being used.

Associated with each page is its *name*, its *age*, and *altered* statistics. The *name* of a page is formed by dividing the absolute storage address by the number of words in the page and is used to determine whether or not a page is resident in the buffer and to form the mass storage address needed to bring a page into the buffer.

The *age* statistic is updated each time a page is referenced and allows the buffering algorithm to determine the least recently used pages.

An *altered* flag is set each time a page is modified and, whenever a page must be purged, the flag indicates whether or not it is necessary to write the page to backing storage.

The difference in speed between references to the buffer and to backing store is so great that the primary factor in influencing speed of simulation is the hit rate, i.e., the percentage of time that the desired word was found within the buffer. The factors controlling hit rate are the number of pages in the buffer, the size of each page, and the characteristics of the program being simulated.

Outstanding features of the paging algorithm include a look-ahead, a pre-paging mechanism to anticipate the direction of the flow of storage references within the program, and also an age-clocking mechanism such that pages which were brought in by the pre-paging algorithm, but were not used within a limited amount of time become candidates for purging. At the time a write back is necessary, an effort is made to find out if other pages that are contiguous on mass storage can be written back at the same time.

5.4.4.2. Processor Simulation

Processor simulation is accomplished by the interpretive execution of the instruction repertoire of the target machine and is controlled by the processor module. This module is responsible for maintaining a simulated GRS, the program address register (P), the addressing environment (PSR, SLR), the interrupt structure, and for controlling the basic machine interpretation loop. Three other FLIT modules are referenced by the processor module:

1. The simulated storage module for the instruction and operand addressing;

2. The I/O control module for the I/O instructions, interrupt and clock handling; and

3. The operator activity through the simulated maintenance panel.

The interpretation of instructions is actually controlled by the operator activity via the simulated maintenance panel. Such maintenance panel features as START, STOP, STEP and inspection and change of control registers and storage are available to the FLIT user to control the simulation process.

FLIT's basic instruction interpretation loop is divided into six parts:

1. Check for Interrupt or Stops
2. Instruction Breakpoint
3. Instruction Fetch/Increment P-Address
4. Instruction Decode
5. Operand Address Calculation/Operand Fetch/Execution/Operand Store
6. Clock Update

The check for interrupts includes requests to stop the processor, requests by the timer module to be called, requests for simulated processor or I/O interrupts, or requests by the I/O module to be called. Stops due to simulated breakpoints and for instruction traces are checked for after the check for interrupts has been completed.

The instruction decode routine isolates the f-, j- and a-fields and uses the result to determine the address within FLIT to call for instruction interpretation. The instruction is further split into the x-, h-, i- and u-fields for the address generation routine.

The majority of instructions are interpreted through a common loop. This loop calls address generation, preloads registers, performs an execute remote of the desired instruction, and finally updates the user's data as the instruction dictates.

The basic processor simulator is controlled by data which reflects the simulated run's requested environment. The machine variables that are maintained by FLIT are:

1. Program Base Register (PSRs)
2. Storage Limits (SLR)
3. D-Bits (PSRs)
4. P-Counter
5. General Register Stack (0-0177)

In system mode, the Program Base Registers (PSRs), Storage Limit Registers and Designator Bits (D-Bits) data are controlled by the privileged instruction set; whereas in the program mode, they are established when the program is loaded, by the MCORE and LCORE ERs, or the LIJ/LDJ instructions. The program address register is incremented once for all instructions, plus an additional incrementation for successful skip instructions, and it is updated for all jumps taken. The general register stack is modified by the simulated instructions, and by FLIT as a result of interrupt simulation.

The basic difference between system mode and program mode is in interrupt processing. In system mode, FLIT modifies the D-bits, saves the PSRs and simulates the instruction at the established MSR setting plus the interrupt type. In program mode, however, FLIT intercepts user-executed ERs and passes them to its ER simulator routine. This routine reconstructs or modifies the user packet (within the FLIT area) to correspond to the simulated environment, and executes the ER via the host Operating System. All other interrupts result in a message and processor stop (i.e., guard mode, privileged instructions, illegal instructions, etc.).

5.4.4.3. Input/Output Control Simulation

The I/O simulation module is responsible for I/O instruction and interrupt simulation and for maintaining the simulated clocks.

When an input/output instruction is encountered by the FLIT processor, the I/O module of the machine activity is entered. It determines the length of time required to complete the I/O transfer, schedules an interrupt to occur at that time in the future, and returns control to the machine activity for other processing. When the clocks show that the time has elapsed, the I/O module regains control again, performs the actual I/O transfer via an I/O Executive Request, and generates the simulated interrupt.

The execution of an I/O instruction, in addition to setting various I/O conditions in FLIT's internal tables, causes the control unit/device simulator module linked to the channel to be called. The device module then acts upon the operation requested by the I/O instruction being executed.

The final function of the I/O simulator is to maintain the dayclock and to create dayclock interrupts.

5.4.4.4. System Clock Simulation

All clocks defined for the machine being simulated are maintained and updated at the end of each simulated instruction execution. The clocks are used to generate simulated dayclock and real-time clock interrupts and to determine when an I/O transfer should occur. The accuracy of the clocks reflect instruction timing only and do not take into account conflicts in main storage and GRS due to other processors, I/O transfers, or instruction and operand fetches. Ignoring conflicts allows for reproducibility of runs and also speeds up the simulation process.

5.4.4.5. Device Control Simulation

Devices, with the exception of the system console, are simulated on mass storage. Using the Operating System file structure, a device may be simulated on its own file or removable device. The method of simulating a device depends upon the device type it is. For tape devices, tapes may be simulated either as physical tapes or on a mass storage file. When simulating the tape as a physical tape, the tape must remain mounted for the duration of the FLIT run, whereas when simulated on a sector-formatted file, a special processor is called which copies the tape to a sector-formatted file in such a format that it can be later simulated as a tape, eliminating the need to have the tape mounted for the simulation period.

All mass storage devices, e.g., FASTRAND III drum, FASTRAND II drum, 8430, 8432, 8434, 8436, 8440, 8460, FH-432 and FH-1782 drum and disk subsystems, are simulated on sector-addressable files. The structure of a file for simulating disk is such that the disk being simulated is prepped in any arbitrary fashion.

Paper peripherals, including the system pagewriter, are simulated on mass storage SDF files by writing to the files using the PRNTA$ ER interfaces.

The device simulation method that has been developed allows devices (both existing or merely in a design stage) to be simulated whether or not actually configured on the host system, and also provides the ability to simulate a maximally configured 1100 Series System.

Communications simulation is not available via FLIT. This capability is currently available using the Communications Simulator (CS1100). (See 5.2.)

5.4.4.6. System Console Simulation

The system console is the only device which is not simulated within the file structure. Instead, it is maintained as a 16x64 matrix within FLIT. The reason for special handling of the console is that it provides for asynchronous conversation between the FLIT user and an executing Executive routine. Therefore, the FLIT user can display the current contents of the "screen", generate an "attention" interrupt and place a message on the screen to be transmitted to the executing Executive routine.

A PAGEWRITER printer, simulated on a mass storage file, can also be configured along with the system console and can be used to provide a hard copy of console output when the FLIT user is in demand mode on a UNISCOPE 100 terminal or other CRT-type device. A hard copy of the PAGEWRITER printer output can be obtained by breakpointing the user file and performing a @SYM operation.

5.4.5. FLIT Input Statements

To control the simulation process, there are two classes of input statements available. These statements are processed internally by the operator activity and include control statements and the operator interface statements.

The control statements or commands are prefixed with an '*', are fixed in format, and provide configuration, action and debug directives to the FLIT processor. The operator interface statements are free form and constitute a high level interactive language that adheres to 1100 Series Assembler Conventions. For details see SPERRY UNIVAC 1100 Series, FLIT Programmer Reference, UP-8435 (current version).

5.4.5.1. Control Statements

The configuration commands include the *CORE and *DEVC statements. These two control statements apply only when operating in the system mode. For the user mode, the FLIT processor controls the simulated environment when the program is loaded. An internal file is maintained for program storage simulation, eliminating the need for the CORE statement. Also, when in the user mode, files used by the program are assigned by the program, and hence, no device configuration statements are required.

*CORE

The *CORE command is used to define the starting and ending main storage address, and a mass storage file that is to be used for simulating the specified main storage area.

*DEVC

The *DEVC control statement is used to configure the peripherals for simulation and provide the information necessary to simulate the device.

5.4.5.2. Action Statements

The action control statements are also submitted by the FLIT operator and provide control over the machine activity by means of the operator activity. These commands represent the actual switches on the maintenance panel that can be activated by an operator when required, and provide the FLIT user with a set of commands which are directly analogous to the hardware functions.

*START

The *START command starts the machine activity and begins the interpretive process.

*STEP N

The 1100 Series hardware instruction step feature is also simulated. When the *STEP command is initiated by the FLIT operator, the machine activity simply executes the number of instructions specified on the command and then stops.

*STOP

The STOP command is a command to stop the machine activity until further input by the FLIT operator.

*LOAD

To load a program into simulated storage, the *LOAD command is available and is used to initialize FLIT for the program mode.

*BOOT

The *BOOT command is used to initialize FLIT for operations in the system mode.

*CRT

The *CRT command is used to simulate the 1100 Series console. This command is only applicable when operating in the system mode and is used to perform operator-type communications with the operating system being simulated.

*DATA

The *DATA command is used to pass data images to a user program in response to a READ$ request executed by a program.

5.4.5.3. Debug Statements

Debug control statements are available to the FLIT user to interrogate the state of the machine and to suspend the simulation process for restart at some later time.

*BRKPT

The 1100 Series breakpoint hardware feature is simulated and permits the breakpoint setting via the *BRKPT command. Relative or absolute breakpoint addresses can be specified along with the type of setting to be initiated (i.e., breakpoint on instruction, reads, writes, etc.).

*TRACE

For debugging purposes, to trace the execution of a program, the *TRACE command allows the user to specify ranges of addresses and conditions under which the instructions within these ranges, when executed, will be displayed symbolically. Instruction mnemonics, contents of registers, and storage locations can be displayed.

*SAVE

The *SAVE command will suspend the simulation process for later processing. When this command is executed, the full configuration, machine and program environments are saved.

When running FLIT, an occasional *SAVE should be done to provide a backup in case an error is encountered and to allow a quicker return when the simulation process is to be restarted.

*RESTORE

In a subsequent FLIT run, the ability to reload a saved environment is provided by the *RESTORE command. Time consuming processing that otherwise would be necessary is eliminated, and simulation processing can begin immediately by the FLIT operator, after the execution of this command.

5.4.5.4. Operator Interface Statements

The other type of input statement previously mentioned is the operator interface statement. These input statements are free in form and constitute a high level interactive language. The operator hierarchy and coding conventions for constants and variables follow that of the 1100 Series Assembler. With these input statements, unlimited debugging capabilities are available to the FLIT user.

The language allows main storage to be treated as a data array, thereby permitting access to allocations via either an octal address or by symbolic name using the diagnostic tables. All simulated internal registers (BR, SLR, P, etc.) and GRS can also be accessed through a set of pre-defined labels. Access implies both the ability to reference and to modify any register or storage location. With access to the diagnostic tables, the language allows for the interrogation and alteration of any aspect of the pseudo machine or program being simulated.

A macro capability is also available that allows the FLIT user to combine sets of control and interface statements to perform logical functions. Among the many capabilities this provides are such things as main storage searches, main storage dumps, elaborate traps, verification of chains, etc., and any others the user may devise. The capability is unlimited in what it can do.

To illustrate the debugging capability that is available via FLIT, consider the situation where a frequently referenced storage location is being overwritten with an invalid value. Attempting to trap this on an actual machine would be quite difficult and time consuming. Under FLIT, however, such a trap is fairly easy. A function can be written which would perform the following steps:

1. Set a breakpoint for any write at the location.

2. Push start and wait for machine activity to stop on breakpoint.

3. Examine storage location. Go to step 2 if value is valid.

4. Inform the user when a bad store is found.

5.5. DEMAND SELECTION BY TIME QUEUE

The goal of a program scheduling algorithm in a demand time sharing environment is to allocate machine resource to demand programs in a manner which was both equitable and responsive. To this end, all demand programs are divided into seven priorities with smaller interactive programs being given the higher priorities and larger and/or non-interactive programs given the lower

priorities. As each demand request is queued, it is time stamped. When it is time to search for storage for a demand request, the first (oldest) entry at each demand priority is examined. A weighted wait time is determined for each request examined. The starting point is the differential time between the selection time and the time at which the request was queued. This time is weighted according to level with higher priority requests being given the larger weight. The request with the longest weighted time is selected. When comparing a higher level request which was just queued with a lower priority request which has been waiting for some time, the lower priority request would be selected for attempted allocation. If the lower priority request is not satisfied, a time will come when the weighting factor will cause the weighted time for the higher priority request to overcome its shorter time on queue. The next pass by the Dynamic Allocator (DA) will select the higher priority request. For any given selection, the weighting function will be followed exactly. The weighting function currently in use corresponds very closely to a power of two level differential.

The weighting function used is:

for TOQ $<$ DTSF1,

$$TOQ_w = TOQ-DTSF1$$

for TOQ $>$ DTSF1,

$$TOQ_w = (TOQ-DTSF1)*DTSF2**N_L$$

where:

TOQ	Time differential between time of selection and time the request was queued.
DTSF1	"Negligible" time factor – currently 1 second.
TOQ_w	Weighted time on queue. The level with the largest TOQ_w will be selected.
DTSF2	Level difference factor. Currently 1.9.
N_L	Number of priorities between the priority for which TOQ is calculated and the lowest demand level.

The actual computation uses integer arithmetic. DTSF1 is expressed in 200 μsec units as are the saved and current times. DTSF2 is defined as multiplied by 64_{10}. It is actually defined as $64_{10}+LDIFF*64_{10}/100_{10}$. LDIFF is then the percentage advantage given to each higher level.

6. Operator Controls

6.1. CHECKPOINT/RESTART

6.1.1. Introduction

Checkpoint/Restart is a mechanism to stop a run or program and create a checkpoint that may be used to restart the run or program at another time.

The Checkpoint/Restart feature is designed for use by long-running programs as a safeguard against time loss due to system failure or shutdown for maintenance, block time, etc. A lengthy program may find it worthwhile to checkpoint itself at intervals so that if the system fails before the program completes, the program may be restarted from the last checkpoint.

The Checkpoint and Restart operations may be initiated by one of three requests:

1. an Executive Request,

2. a control statement, or

3. an unsolicited console keyin.

The program with a run time of several hours might include an ER type checkpoint every half-hour, while an operator would utilize the unsolicited keyin to checkpoint all active batch runs prior to a change of schedule. For further details see SPERRY UNIVAC 1100 Series Executive System, Volume 2, EXEC Programmer Reference UP-4144.2 (current version).

6.1.2. EXEC Level Compatibility

There are some EXEC levels for which Checkpoint/Restart is not upward or downward compatible. That is, a checkpoint taken on one level cannot be restarted on either an earlier or later level. This is due to changes in the checkpoint mechanism or to changes in EXEC table formats or procedures.

6.1.3. Security

The need for security of the checkpoint file itself cannot be over emphasized. Many safeguards have been added, however, all possible precautions should be taken to provide security for this file.

6.1.4. Print File

The user's print file is not saved as part of the complete checkpoint. When a restart is requested, the print file is re-initialized at the point where the checkpoint was taken. No attempt is made to recover any print developed prior to the checkpoint call. Therefore, it may be advisable to do a @BRKPT prior to internal checkpoint calls in the program.

6.1.5. Unsolicited Keyins

The unsolicited keyin (CK) may be used to checkpoint a batch run and must be taken to a tape file. If successive checkpoints are keyed in for the same file, the requests are queued and the file remains assigned to the Executive until all requests are satisfied, at which time the operator is given the option of releasing the file or retaining it for further use. This is also true of the Restart (RS) keyin. Successive RS keyins need not be made in the same order in which the checkpoints exist on the file, since a rewind of the tape and a search for the specified checkpoint is done for each request. The tape then remains positioned at the end-of-file mark following the checkpoint which was just restarted. For that reason, once a restart has been attempted, no additional CK keyins must be made to that file since checkpoint begins writing at the file's current position, and will destroy any checkpoints written further out on that tape. Also, RS keyins must not be requested from a file until all checkpoints to be taken to that file are completed.

Following completion of any CK or RS operation, the checkpoint file can only be released by the operator's 'Y' response to the 'SHOULD filename BE RELEASED' message. This must be done following the last intended use of the file in order to free the tape unit for other system uses.

6.2. EXEC SETTING OF THE BREAKPOINT REGISTER

This feature may be used to control the usage of the P-trace feature. Also, if system failures occur due to a specific main storage location being overwritten, a BP keyin may be applied to catch the offending code. Occurrence of the breakpoint interrupt for the EXEC results in an EXERR 0160.

The breakpoint register may be set from the on site console with a 'BP' keyin. Several formats are allowed so that the force equality designators and W, R, P and S bits may be set. The keyin is:

BP *address* sprw mask*

where:

address	The address (relative or absolute) of the BPR setting.
*	Optional – indicates address is absolute.
sprw	The Fieldata representation of the S, P, R and W bits. These may be entered in any permutation and/or combination. The'B' bit of the BPR will always be set.
mask	Two octal digits to specify equality comparison ofcorresponding bits of specified address.

The only format requirements of the keyin are that each field (with the exception of *address** must be separated by at least one space. The *mask* and *sprw* fields are interchangeable.

A KEY ERROR message will be returned if the address is out of EXEC range, *mask* is a value greater than can be expressed in six bits or the *sprw* field contains some invalid Fieldata letter, or 'PWR' conflicts (bits 32, 31, 30).

In addition to setting an EXEC breakpoint, the BP keyin also has the following applications:

1. BP *(transmit)* – Display current breakpoint register setting.

2. BP 0 *(transmit)* – Clear the breakpoint register.

6.3. SYMBIONT CONTROL KEYINS

The symbiont control keyins can be divided into two classes, those that control the operation of symbiont devices and those that manipulate symbiont files.

The SM and GO keyins are used to control the operation of such symbiont devices as card readers, printers, punches and remote terminals. The SM keyin can be used to force a device to discontinue processing a file and to proceed with the next file; initiate or lock out devices; reprint, repunch or advance without printing or punching; and suspend or terminate a device.

The GO keyin causes activation of a remote site by the central site operator. It provides a means of activating receive-only terminals which have no means of transmitting the sign-on message.

The SQ, SX, SV and SR keyins are used to manipulate the queued symbiont files, that is, those files which are ready to be printed or punched. These files may be the standard PRINT$, PUNCH$ or user files with qualifier*filename(f-cycle) specifications. These symbiont files are directed to specific printers or punches (onsite or remote) or to a group which comprises several devices.

The SQ keyin can be used by the operator to obtain information about the symbiont files that are currently queued for output. The information can be displayed as a general status for all output devices, status of a specific device or group, and status by specific runid and filename. The information displayed provides the operator with number of files queued, filenames of those queued, page and card size of files, and priority of the files in the queues.

Additional SQ keyin functions allow the operator to manipulate the files displayed on the status keyins. The operator can change the priority of a file or set of files and thus can force the files to be output either before or after other files currently queued to be output.

The SQ keyin can also be used to redirect queued output files to a different device or device group. This redirection can be done to a single file or to all files queued to a device. This feature can be used when output devices become available or unavailable or to provide better distribution of output files to devices. If a device is to be permanently locked out, for instance, a means is provided by which all future files that would have been queued to that device will be automatically redirected to another available device.

Another symbiont keyin, SX, can be used to delete symbiont output files from the queues and thus prevent the output of the files. As with other symbiont keyins, this function can be performed on a specific file or sets of files.

A pair of symbiont keyins, SV and SR, are available to save to tape and then, at a later time, restore a set of symbiont output files which have been queued. These files, once placed on tape, are no longer on the system symbiont output queues. These functions can be used to save large quantities of output files during peak production time and then later restore them for output during non-peak production time when the load on output devices is reduced. They can also be used to move files from one system to another for output.

7. The File System

7.1. MASTER FILE DIRECTORY

7.1.1. Master File Directory (MFD) Structure

For catalogued files, entries are constructed containing the identification and characteristics of each file, and these entries are maintained by the system in the MFD. The MFD consists of look-up table entries and directory items. A look-up table is linked to the catalogued files. Each filename and qualifier are folded to provide an index into the look-up table. The length of the look-up table is a system generation parameter. Directory items are 28-word areas used to store information needed to maintain a file's identity and description. Directory items are defined as:

- Search Item

  Used to locate pointers to lead items which have the same index into the look-up table.

- Lead Item

  Used to provide the link between the qualifier/filename combination and each file of the F-cycle set. The lead item contains pointers to the main items.

- Main Item

  Used to store information which defines a file. There is one main item for each file of the F-cycle set. The main item contains a pointer to the granule item.

- Device Area Descriptor (DAD) Table/Reel Number Table

  A DAD table is comprised of entries which correlate the file relative location of the file's contents with the physical location of the file's contents. A reel number table contains a list of reel numbers assigned to a catalogued tape file.

A word in the look-up table contains one of the following types of entries:

- Zero

  A look-up table word is zero if no file set has an index equal to the word number.

- Pointer to Lead Item

  If only one file set has an index equal to the word number, the look-up table word points to the file set's lead item.

- Pointer to Search Item

  If more than one file set has an index equal to the word number, the look-up table word points to the search item.

Figure 7-1 illustrates an MFD entry.

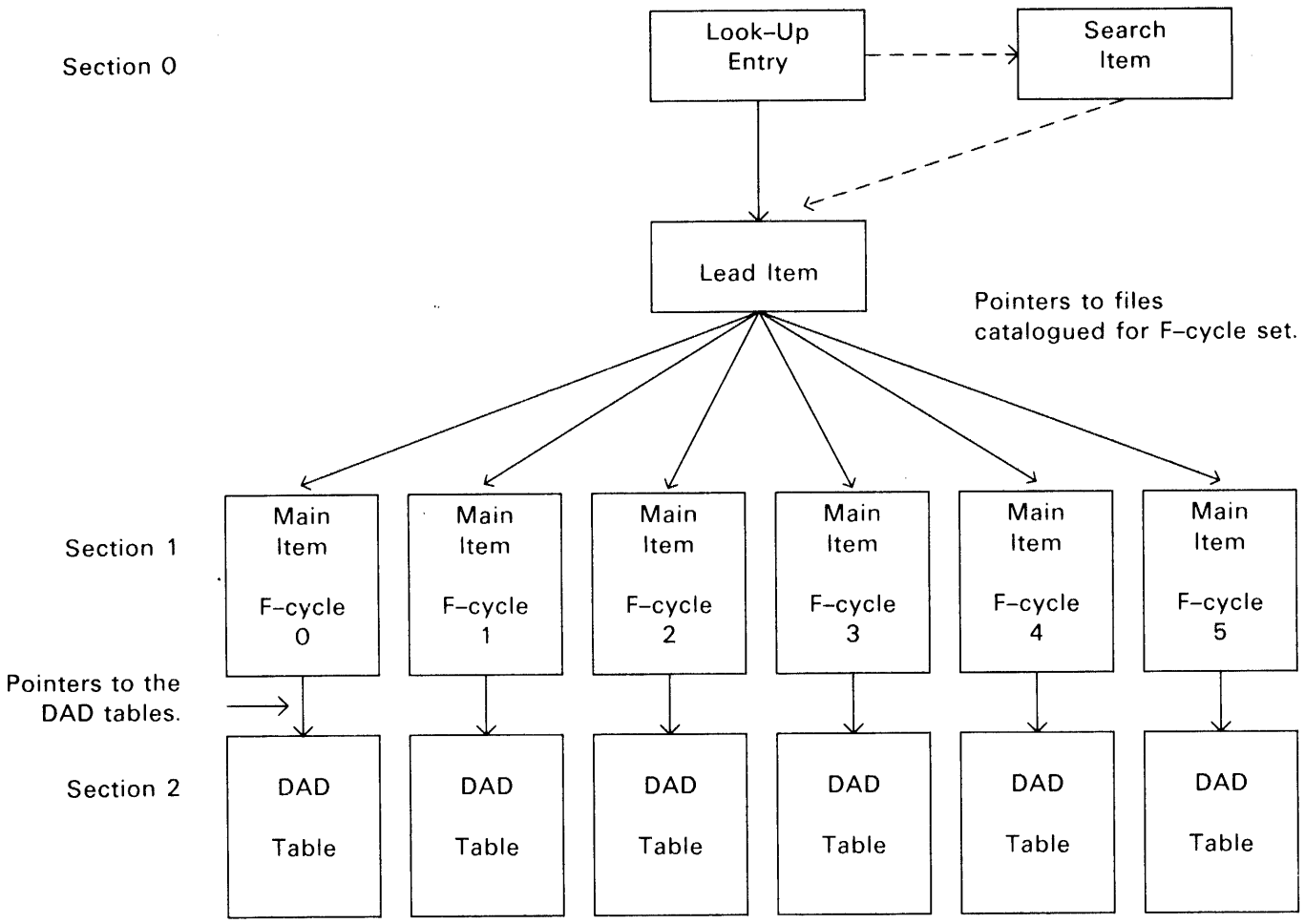


Figure 7-1. Example of a MFD Entry

7.1.1.1. Mass Storage File Addressing

A primary object of the MFD is to provide the user a convenient means to access a mass storage file. Different mass storage devices have unique addressing methods, a wide range of storage capacities and other significant differences at the hardware level. In addition, the Executive determines file placement according to current mass storage availability and a "best fit" algorithm, among other parameters. In order to unburden the user from needing to know where a file is physically placed and how to access that particular device, a file relative addressing mechanism is utilized.

The user accesses a file by referencing a filename and a location relative to the start of the file without regard to where the file text physically resides. The Executive determines the actual physical location of the file by using the Device Area Descriptor (DAD) tables. The primary unit of reference is the three word DAD entry, which describes a length of file space. File space described by a DAD can either represent allocated physical storage area on some device, or an expanse of unallocated file area which exists in the file's addressing structure, but for which no physical storage space has been allocated. This unallocated file space is commonly referred to as a 'hole'. Current Executive philosophy holds that unless otherwise directed, if a user writes into the first track of a file and the twelfth track, there is no need for the intervening ten tracks to be represented by allocated mass storage space. A DAD representing ten tracks of unallocated file space maintains the file relative addressing structure without utilizing costly mass storage space. It is also true that contiguous file relative mass storage space need not be physically contiguous. It is possible for tracks one, two and three of a file to be allocated on three different mass storage devices, and the user program can access these tracks with contiguous file relative addresses.

In order to accomplish this file relative to physical address transformation, it is necessary to be able to describe a point on mass storage with a concise notation, namely by using the Logical Device Address Table (LDAT) index and the device relative address.

7.1.1.2. Logical Device Address Table (LDAT)

LDAT is a resident Executive table which provides the logical ordering of all mass storage devices configured in the system. The Executive uses this ordering to address devices by a logical index.

The LDAT format is shown in Table 7-1.

Table 7-1. Logical Device Address Table (LDAT)

0	*unused*	*Length of LDAT*
1	*address of the unit status table associated with this LDAT index*	*address of the logical device unit status table associated with this LDAT index*
2		
n-1		
n		

7.1.1.3. Device Relative Address

A device relative address is a specific location on a device expressed in words. The first addressable word on the device is word 0 and all succeeding words follow in contiguous order.

7.1.1.4. Device Area Descriptor (DAD) Tables

The DAD tables are used by the Executive to correlate file relative addresses with physical mass storage addresses. Every mass storage file has at least one DAD table (see Table 7-2). On every I/O reference to a file, the Executive uses the file relative address as an index into a DAD table chain to find the physical location of the file relative address.

The primary unit of reference in a DAD table is the three word DAD entry. A DAD is composed of four fields:

- Device Relative Address

  Defined in 7.1.1.3.

- Number of Words

  The number of words in a contiguous area.

- Flag bits

  Descriptors that define characteristics of the area.

- Device Index

  Specifies the device where the area resides.

Table 7-2. Device Area Descriptor (DAD)

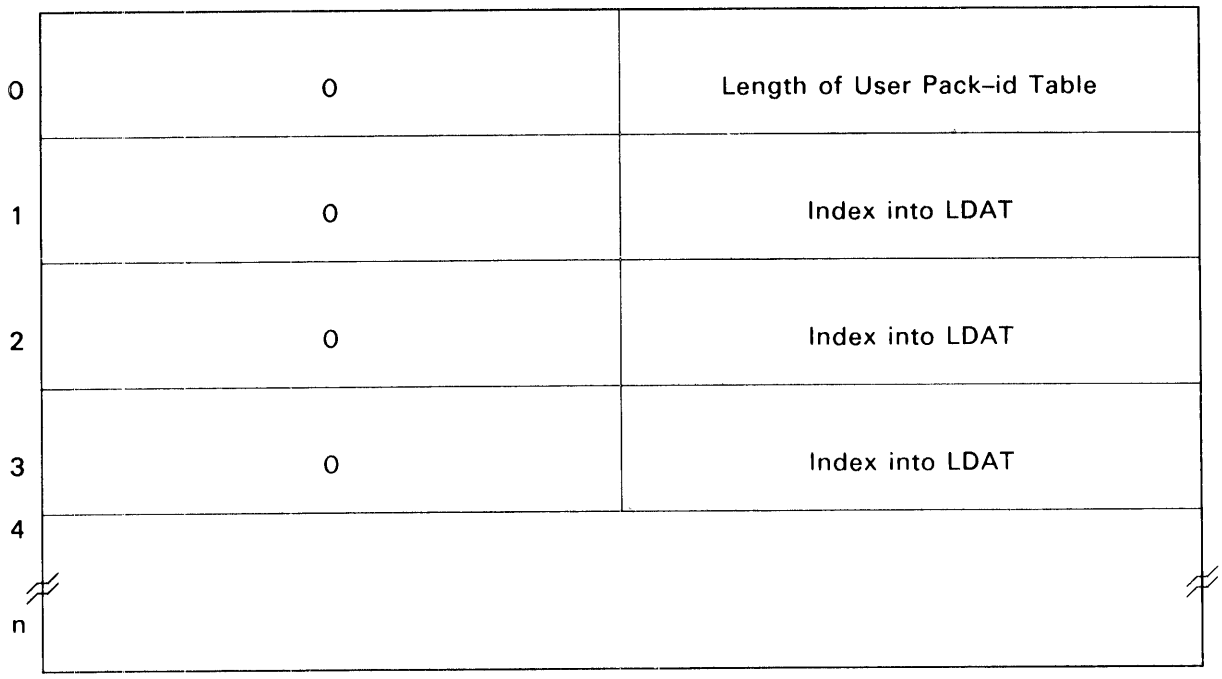
Device Relative Address	
Number of Contiguous Words	
Flag Bits	Device Index

All addresses and lengths are specified in words. This removes equipment type dependencies and simplifies and shortens code that utilizes those fields.

7.1.1.4.1. Device Index

For fixed mass storage, the device index is an index into the Logical Device Address Table (LDAT). For removable disk mass storage, the device index is an index into the user pack-id table. A negative value in the device index field signifies an unallocated area in the file relative addressing structure.

The format of the user pack-id is as follows:



Thus, it can be seen that for removable disk, the device index points indirectly to LDAT, via the user pack-id table. The user pack-id table resides in the Program Control Table (PCT) and contains one word for each pack assigned to the user file.

7.1.1.4.2. Flag Bits

The flag bits in a DAD are defined as follows:

Bit 18	Designates a dynamically detected bad spot.
Bit 19	Removable disk bit.
Bit 20	End of DAD table.
Bits 21-35	Reserved for future use.

7.1.1.4.3. DAD Tables

A DAD table contains all the information necessary to convert a file relative address to an absolute address. Two range words define the upper and lower limits of the file relative addresses described by the table. One of these range words gives the file relative address of the first word in the table while the other range word gives the file relative address of the last word described in the table (biased upward by one for ease of computation). Up to eight DAD entries per table define the physical location of the file space. Since each DAD entry contains the number of words described by the DAD, it is a simple task to find any file relative address. First do a range check on a file relative word address to find the correct table, then add the number of words in each DAD to the starting range word of the table until the DAD entry containing the file relative address is found.

DAD tables also contain forward and backward directory links to logically contiguous DAD tables. In addition, when a DAD table is kept in main storage while a file is assigned, a forward link to the next DAD table in main storage is maintained, along with a flag that indicates whether the DAD table has been changed since it was last written out. Table 7–3 shows the DAD table.

7.1.1.4.4. Mass Storage Allocation

Mass storage is allocated in granules of either 03400 words (track granularity) or 0340,000 words (position granularity).

Mass storage is allocated to a file as a result of an initial reserve request on an @ASG statement or as a result of a write into a previously unallocated granule of a file.

As a general rule, the Executive tries to allocate physically contiguous space for a logically contiguous file area. However this is often impossible with a dynamically changing allocation scheme such as the Executive uses. This is especially true of files which have no initial reserve, where space is allocated dynamically as a user program writes into the file. A three word DAD describes a mass storage area that can vary in size from one granule to an enormous number of granules. It is naturally more efficient for granules to be allocated contiguously, both in terms of directory space saved by using fewer DADs and in terms of the amount of computation needed to convert a file relative address to a physical address. For this reason, it is more efficient to use the initial reserve field on the @ASG statement, especially where the size of the file is known and where most of the file would consist of logically contiguous data.

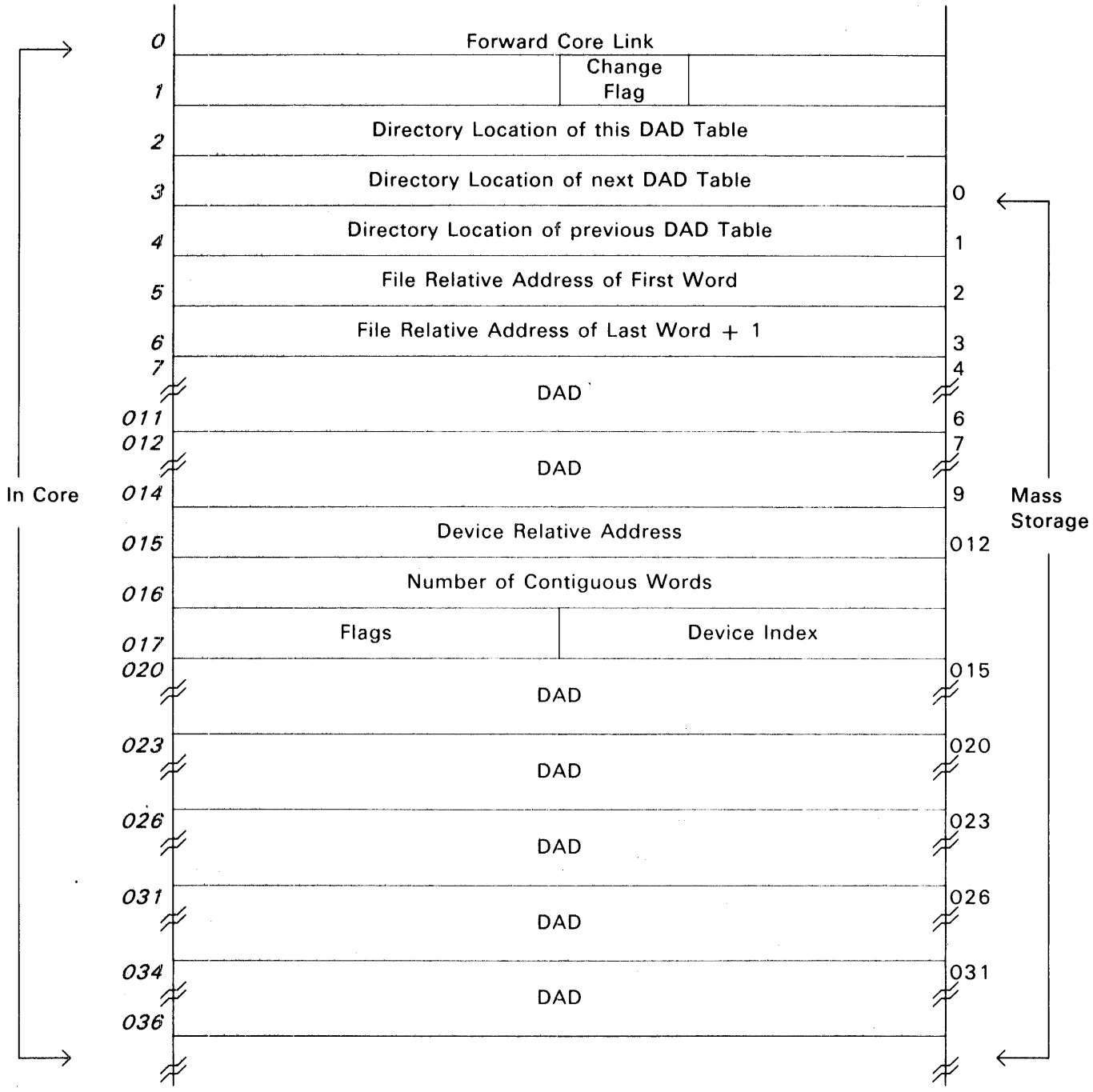
7.1.1.4.5. Master Bit Table (MBT)

The MBT (Table 7–4) is a bit map of mass storage space on a unit. Each bit represents one FASTRAND drum formatted track. If a bit is set, the track is allocated. Each word represents 32 tracks. Two words represent one sector formatted position. Bit 35 represents the lowest numbered track, bit 4 is the highest numbered track in the word. The bottom four bits in each word do not represent any mass storage space. A checksum feature is available and when configured the last word in the MBT contains the checksum value. The length of the MBT is variable and depends upon the size of the device. If the available space on a device does not end on a position boundary, the bits representing nonexistent space in the last position are set as allocated when the MBT is initialized.

If a high speed drum is used as the system drum, MBTs for all drum and FASTRAND drum type devices are placed on the system drum. Otherwise, they are located on the first directory track of the device.

For disk type devices, MBTs are always kept on the first directory track. There are two MBTs for each disk type device. There is a hardware MBT and a software MBT. The hardware MBT is built when a disk is prepped. All defective tracks are marked as allocated in the hardware MBT. When a disk is initialized or recovered, the hardware MBT is used as a template for the software MBT so that defective tracks can never be allocated to user files.

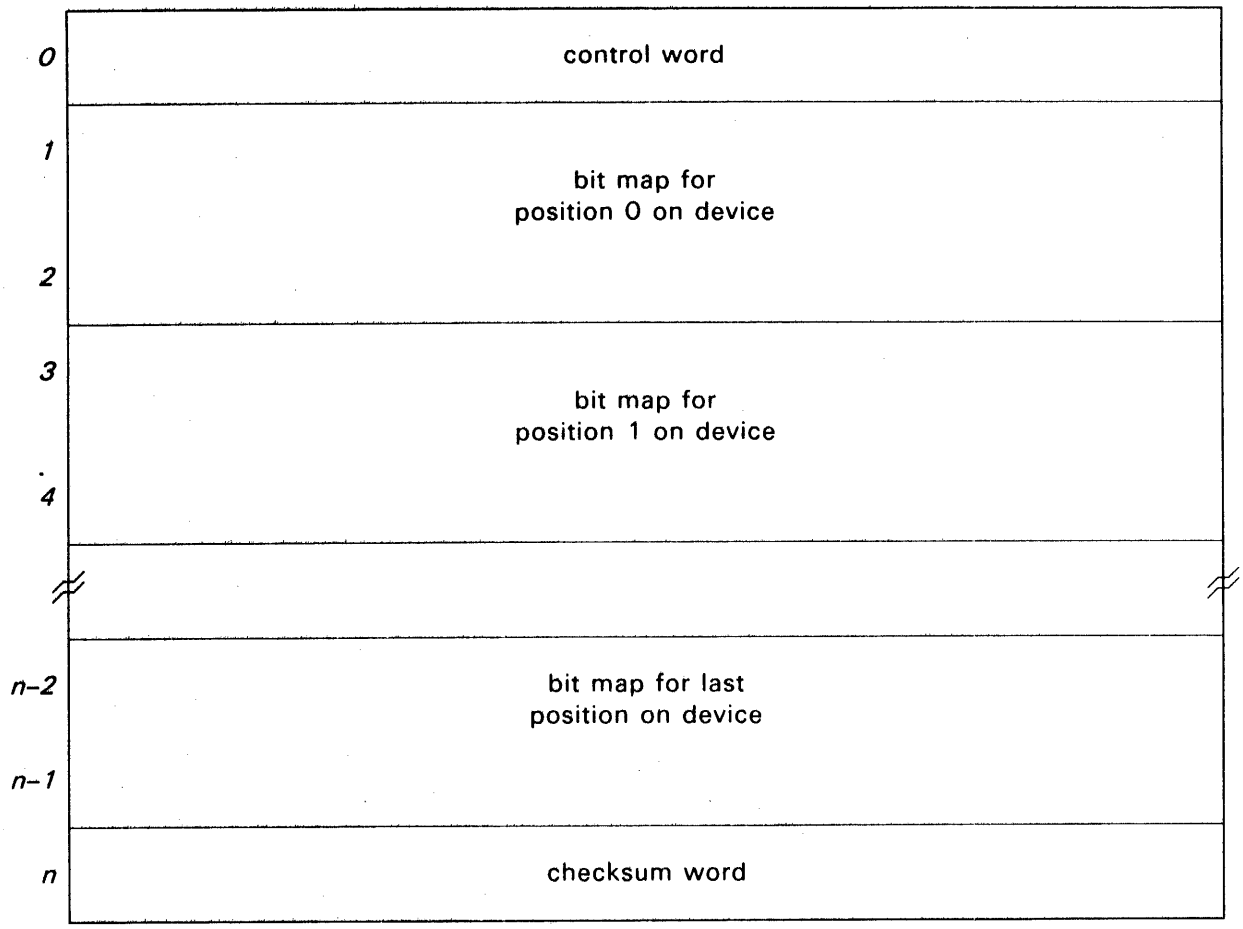
Table 7-3. Device Area Descriptor Table



NOTE:

Bit 33 word 4 is new format flag (NFF).

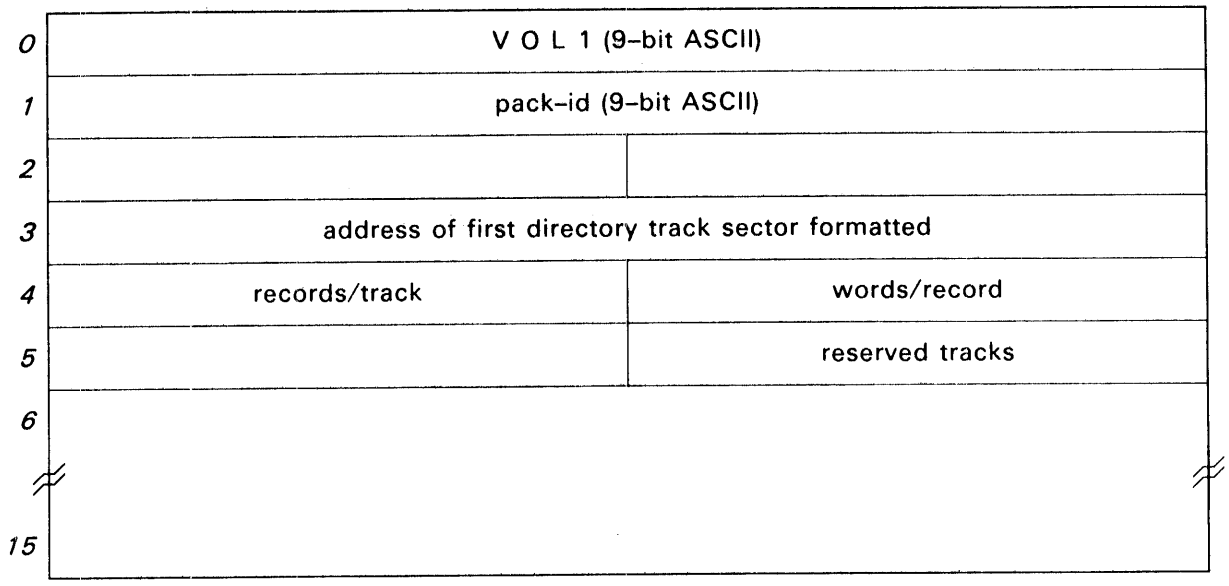
Table 7-4. Master Bit Table



7.1.1.4.6. Tables Unique to Disk Type Devices

Disk type devices have two tables utilized by the file recovery and pack verification segments of the Executive which do not exist on nonremovable storage such as high speed drums or FASTRAND drum devices. These tables, the VOL1 label block (Table 7-5) and sector 1 of the first directory track (Table 7-6), are used by the Executive to identify the disk pack and to locate key pieces of directory information. The tables are unique to disk type devices due to the unique problems presented by removable mass storage.

Table 7-5. Standard Disk Label Format



VOL1

This is the disk's label block. It contains the pack-id and prep factor of the pack. It also contains the location of the first directory track on the pack. VOL1 is in the same fixed location on every pack.

Table 7-6. Sector 1

0	address of hardware bit map		
1	address of software bit map		
2	max (initial) track availability	max (initial) position availability	
3	current track availability	current whole position availability	
4	Fieldata pack-id		
5	LDAT index (0 = removable pack)	bit map length	
6	A		keyid
7	time stamp for pack registration		
010	records/track		words/record
011	reserved for future use		
027			
030	private pack key		
033			

Sector 1

Located in Sector 1 of the first directory track, this table contains the following information:

- Mass storage availability information
- Location of the hardware MBT and software MBT
- LDAT index of the pack if it is a fixed type pack. This LDAT index is assigned at the time the pack is initialized and stays unique to this pack through all subsequent recovery boots. If this field is zero, the pack is a removable disk pack.
- A – Word 6, bit 34: Activity Flag used for removable packs. If set, it indicates that at least one file on the pack is currently assigned to a user run.
- A time stamp showing the last time a removable pack was registered. This is utilized by the automatic pack verification routine.

7.1.1.4.7. Mass Storage ID Table

The Mass Storage ID (MSID) table exists within the Master Configuration Table (MCT) and is used on recovery boots to determine if uninitialized mass storage devices are being added to the system. The MSID table has a one word entry for every mass storage device in the system. This one word entry is indexed to by the LDAT index of the device. The word contains the logical device name of the device, or, for fixed disk units, contains the pack–id of the fixed pack carrying that LDAT index. The MSID table is used on recovery boots to guarantee that the indexing structure is rebuilt accurately.

On tape recovery boots, the MCT is not recovered and the MSID table is lost. Therefore serious mass storage problems can result from modifying the mass storage configuration while doing a tape recovery boot. When doing a tape recovery boot, be sure that no mass storage devices are added to the configuration either through a change in SYSGEN parameters or by using UP keyins to include previously downed units.

7.1.1.5. MFD Structure

The MFD consists of many linked directory sectors. These sectors are maintained on mass storage within track size buffers referred to as directory tracks. A file's directory information is maintained on the same device as the file's data. This is done to safeguard against total MFD loss upon a device failure. Hence, at least one directory track exists for every mass storage device.

A directory track is allocated in sector increments as required. When all 64 sectors of a directory track are allocated another track is acquired. The list of the acquired directory tracks and a map of allocated directory sectors within each directory track is maintained within a mass storage sector called the Directory Allocation Sector (DAS).

A DAS will hold information for up to nine directory tracks. One word is used to hold the track address and two words are used for a 64–sector bit map for each directory track. The last word is used as a forward link address to the next DAS for the device if required. Hence every ninth directory track on a device starting with track zero has a DAS in sector zero.

The format of a DAS within a directory track is:

Table 7-7. Directory Allocation Sector (DAS)

	35	3 0
0	*LDAT-index*	
1	*allocation-bits-for-first-32-sectors*	unused
2	*allocation-bits-for-last-32-sectors*	unused
3	S *MFD-track-addr*	
4	*allocation-bits-for-first-32-sectors*	unused
5	*allocation-bits-for-last-32-sectors*	unused
6	*up to seven three-word entries same as words 3 thru 5*	
26		
27	S *link-to-next-DAS*	
28	*MFD items (see 7.1.2 for MFD item formats)*	
1791		

NOTE:

Throughout this section the word 'unused' implies that the field so designated is not guaranteed to contain or maintain a specific value.

Word 0

Bit 33 used for New Format Flag (NFF).

Word 1

Allocation bits for the first 32 sectors of this track. Bit 35 corresponds to sector 0, bit 34 to sector 1, and so on. The bit is set if the corresponding sector is allocated.

Word 2

Allocation bits for the last 32 sectors of this track. Bit 35 corresponds to sector 32, bit 34 to sector 33, and so on. The bit is set if the corresponding sector is allocated.

Word 3

Address of the next MFD track. If this word is negative (S = 1), then an MFD track does not exist for this entry. This track does not contain a DAS in sector 0.

Word 4

Same as word 1.

Word 5

Same as word 2

Word 27

Address of the next MFD track with a DAS in sector 0. If this is negative, there are no more MFD tracks for this unit.

7.1.1.5.1. Removable Disk

The directory for each removable disk pack is a separate entity. The main difference between a removable disk directory and the MFD is that the removable disk directory has no look up table, no search items and no lead items. These tables are not required because file assignment is not accomplished through direct use of the pack directory.

An Executive automatic pack registration routine registers the files contained on a removable pack when the pack is mounted. The Executive uses the directory information on the pack to create directory items for all the files on the pack in the fixed storage directory. After a pack is registered, the fixed storage directory is used to control any references by user runs to files on the pack.

7.1.1.6. General Description of the MFD File

The directory file is a temporary, sector addressable file assigned to the Executive. The file spans all mass storage units, with at least one track allocated on each unit. The logical addresses of the file are partitioned into ranges of octal 10,000 tracks each, with a range of addresses dedicated to each mass storage device in the system. The file relative addresses of all directory tracks on a unit fall within this range. Therefore, it is possible to look at a file relative directory item address and determine which physical unit the address resides on; and units can be added or removed without upsetting the logical addressing structure of the file.

7.1.1.6.1. DAS--Relative Addresses

A DAS–relative address is equal to the depth of a directory track into a DAS table chain. Thus, the first directory track on a device is DAS–relative track 0 on the device, the track contained in the fourth track address cell in a DAS is DAS–relative track 3, and so on.

Directory item addresses are derived from the DAS–relative location of the track being allocated from, and the LDAT index of the unit being allocated on.

7.1.1.6.2. File Relative Address

Directory item addresses are file relative addresses. A directory address is derived from the LDAT index of the unit being allocated on, the DAS-relative location of the track being allocated on, and the sector number within the track.

A file relative directory link address format is:

S1	S2 & S3	S4 & S5	S6
descriptor bits	Unit-separator (LDAT index)	DAS-relative-directory-track	sector number

where:

S1 of the directory link is used for descriptor bits, which are described in 7.1.2.

Bits 29 - 18 (S2 & S3) indicate which unit the file relative address is on. This portion of the file relative address is called the Unit Separator, and the value in this field corresponds to the LDAT index of the device.

Bits 17 - 6 (S4 & S5) describe the DAS-relative directory track within the addressing range of octal 10,000 tracks dedicated to the unit.

Bits 5 - 0 (S6) describe the sector number within the track.

Thus, all the sector addresses from octal 1,000,000 to 1,777,777 describe unit 1; the addresses from octal 2,000,000 to 2,777,777 describe unit 2; and so on.

The addressing range of the directory file is then:

4096 directory tracks per unit.

4095 mass storage units per system.

7.1.1.6.3. Directory Link Addresses on Removable Disk

The portion of the directory item addresses containing the LDAT index is zero for removable disk directory items. The LDAT index of the unit being referenced is added to H1 of the directory address before an I/O request is initiated. The correct LDAT index is retrieved from the user pack-id table.

It is useful to keep in mind that the directory file is not a removable disk file. The directory space on a removable disk is dynamically included in the directory file as fixed storage space while the removable disk is online.

7.1.2. Directory Item Formats

The MFD contains directory items for each catalogued file in the system. The content and format of the MFD are subject to change without prior notice. The directory item description is as follows:

Bits 35 through 32 in word zero of each directory item have the following significance:

Bit	Description
35	When set to 0, indicates that word 0 of this sector contains a link address to the next sector.
34–32	001_2 – Search item
	010_2 – Lead item
	100_2 – Main item
	000_2 – DAD tables, or sectors 1–n of main item, or sector 1 of lead item. (See Tables 7–8 through 7–16 for examples of these items.)

Table 7–8. Search Item

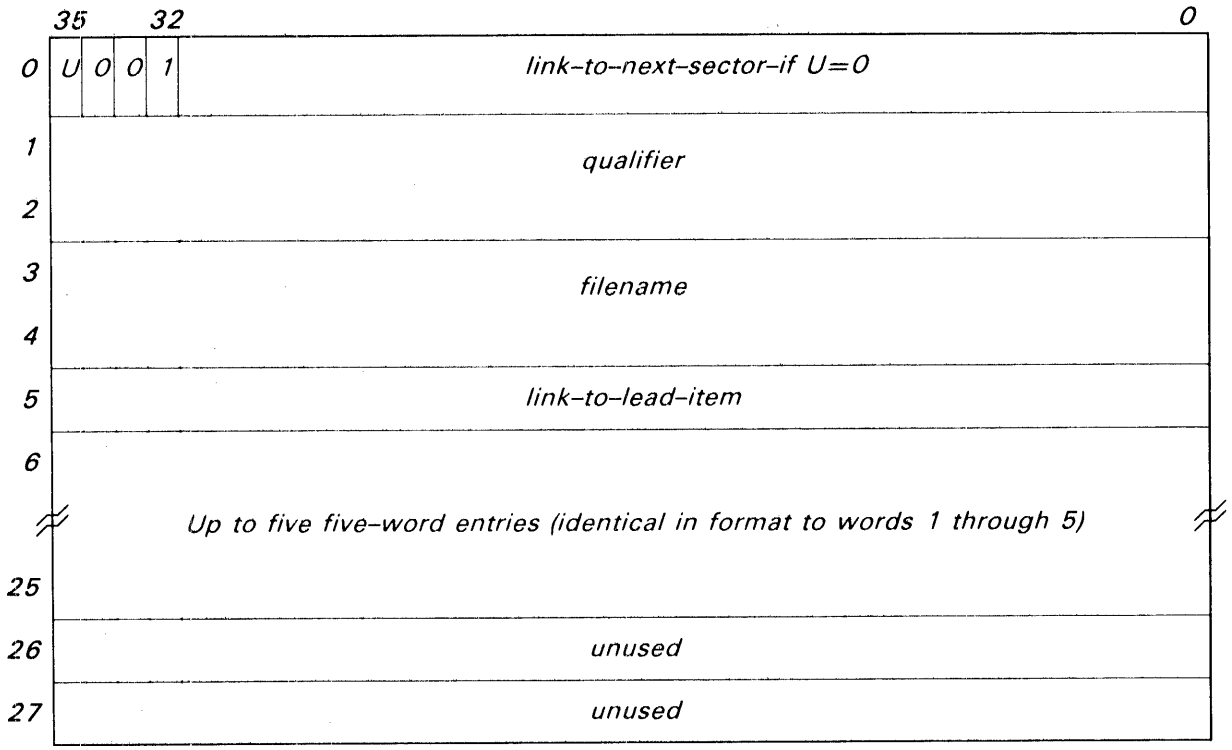
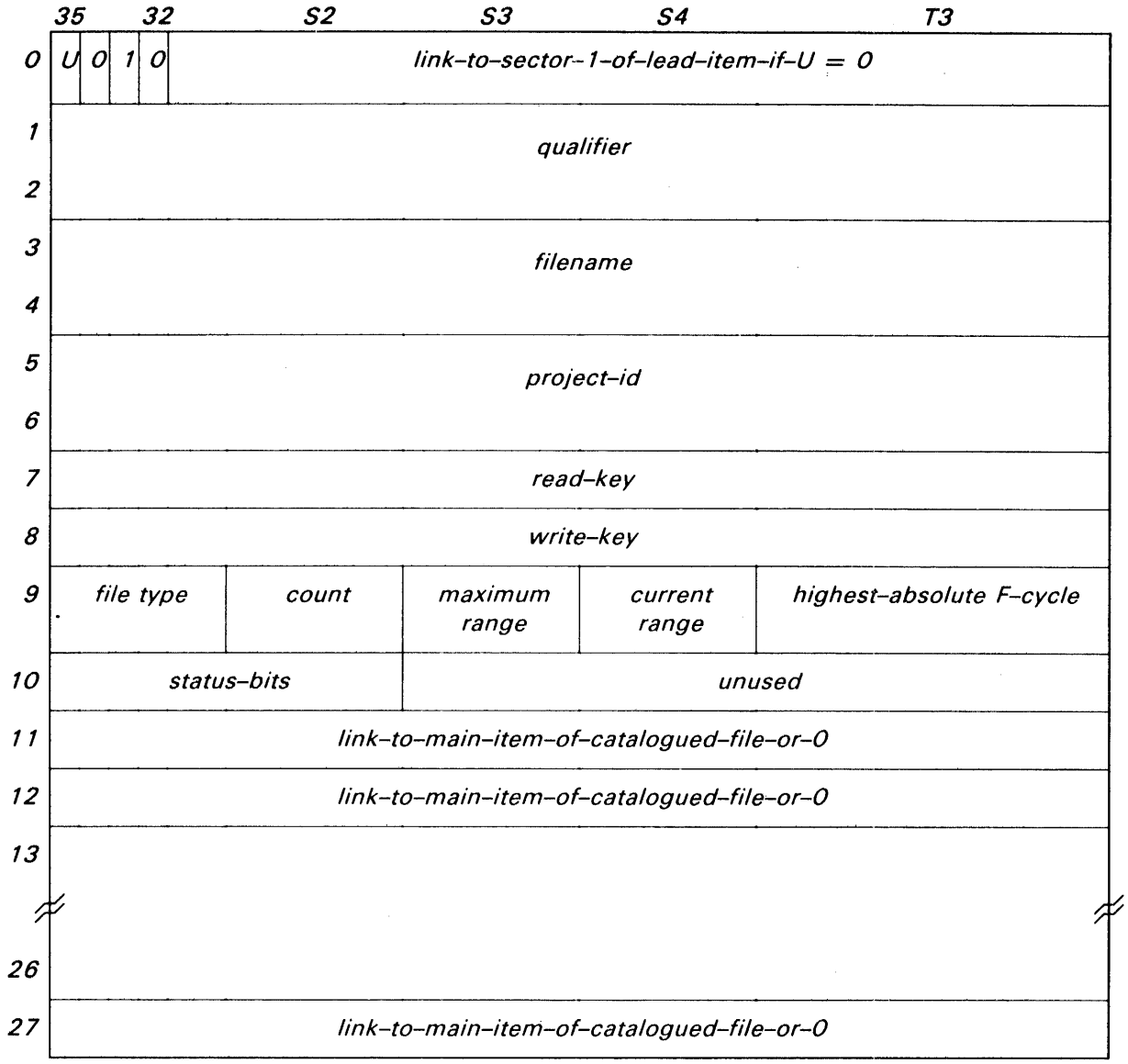


Table 7-9. Lead Item – Sector 0



Word 9

file type — Type of device which F-cycles describe:

000	Mass storage
001	Magnetic tape
040	Removable disk

count — Current number of F-cycles in the lead item.

maximum-range — Maximum number of F-cycles permitted for the file.

current-range — Range of absolute F-cycles currently in this lead item; (highest-absolute-F-cycle)—(lowest-absolute-F-cycle) + 1.

highest-absolute F-cycle — The absolute F-cycle number whose link is in word 11 or would be in word 11 if it existed.

Word 10

status-bits — These bits, when set, indicate:

35	@ASG,G	Guarded file
34	A (+1) F-cycle currently exists (link in word 11)	
33-30	Reserved	
29	Absolute maximum F-cycle	
28-24	Maximum F-cycles in lead item sector 0	
24	First link in lead item extension	
24-27	Maximum F-cycles in lead item sector 1	

Word 11

Link to the main item which describes the highest-absolute-F-cycle. If this absolute cycle does not exist, the entry is 0.

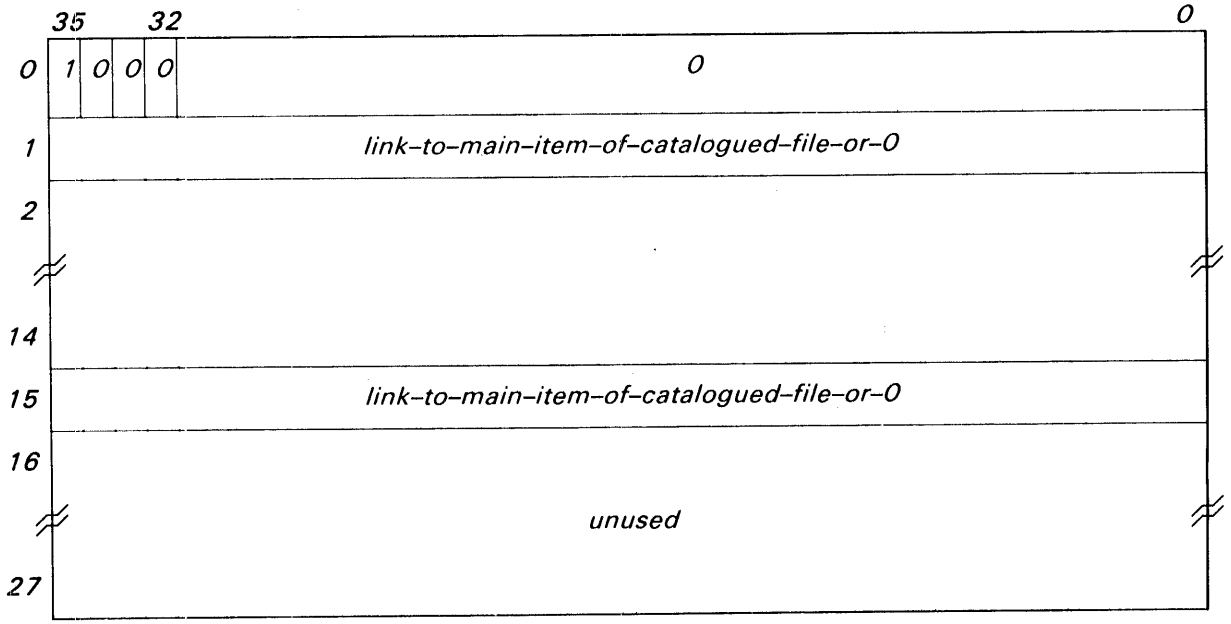
Word 12

Link to the main item which describes a file whose absolute F-cycle number is one less than the highest-absolute-F-cycle. If this absolute F-cycle does not exist, the entry is 0.

Word 27

Link to the main item which describes a file whose absolute F-cycle number is 16 less than the highest-absolute-F-cycle. If this absolute F-cycle does not exist, the entry is 0.

Table 7-10. Lead Item - Sector 1



Word 1

Link to the main item which describes a file whose absolute F-cycle number is 17 less than the highest-absolute-F-cycle. If this absolute F-cycle does not exist, the entry is 0.

Word 15

Link to the main item which describes a file whose absolute F-cycle number is 31 less than the highest-absolute-F-cycle. If this absolute F-cycle does not exist, the entry is 0.

Table 7-11. Mass Storage File Main Item - Sector 0

	35	32	29	23	17	11	0
0	U	1	0	0	link-to-DAD-table-if-U = 0		
1	qualifier						
2							
3	filename						
4							
5	project-id						
6							
7	account-nbr						
8							
9	block-size (negative if variable)				item-size (zero if item is variable in length)		
10	time-of-first-write-following-unload-or-backup-item (TDATE$ format)						
11	disable flags	link-to-lead-item					
12	descriptor-flags		block-buffering-EOF-sector-addr				
13	PCHAR flags	link-to-sector-1-of-main-item					
14	assign mnemonic						
15	symbiont-link-to-initial-SMOQUE-entry				total-nbr-of-times-this file-has-been-assigned		
16	run-id-or-EXPOOL-addr						
17	unused	inhibit flags	nbr-of-runs-currently assigned-to-this-F-cycle		absolute-F-cycle-nbr		

Table 7-11. Mass Storage File Main Item – Sector 0 (continued)

18	*date-and-time-current-assignment-started-or last-assignment-ended (TDATE$ format)*	
19	*date-and-time-of-cataloguing (TDATE$ format)*	
20	*initial-nbr-of-granules reserved-for-this-assignment*	*nbr-of-granules-of-Quota-Group-1*
21	*maximum-nbr-of-granules*	*nbr-of-granules-of-Quota-Group-2*
22	*highest-granule-nbr-assigned*	*nbr-of-granules-of-Quota-Group-3*
23	*highest-track-written*	*nbr-of-granules-of-Quota-Group-4*
24	*unused*	*nbr-of-granules-of-Quota-Group-5*
25	*unused*	*nbr-of-granules-of-Quota-Group-6*
26	*unused*	*nbr-of-granules-of-Quota-Group-7*
27	*user-unit-selection-indicators*	*nbr-of-granules-of-Quota-Group-8*

Word 9

block-size — Used by block buffering package

item-size — Used by item handler

Word 10

time-of-first-write following-unload or-backup-time — If word 12, bit 35 = 1, then word 10 contains the unload time. If word 12, bit 35 = 0, then word 10 contains the time of the first write after the backup file was created or all zeros.

Word 11

disable-flags — Bits 35 through 32, when nonzero, have the following significance:

- 1100_2 Facilities reject (file disabled because it was destroyed)
- 1010_2 Facilities warning (file disabled because of an incomplete write)
- 1001_2 SECURE processor reject (file disabled because it was rejected by the SECURE processor)

Word 12

descriptor-flags — These bits, when set, indicate:

Bit	
35	Unloaded
34	Backed up
33	Unused
32	Has lapse entries
31	Bad main item sector 1
30	Old item format (created on Executive Level 32 or earlier)
29	Tape file
28	Communication between recovery and MFD maintenance routines for removable disk files. Extension sectors of the main item may have been changed on permanent storage. This assures the subsequent update of extension sectors on the disk packs.
27	Removable disk file
26	File is to be made write-only
25	File is to be made read-only
24	File is to be dropped

Word 13

PCHAR-flags — These bits, when set, indicate:

Bit	
35	Position granularity
34	Not used
33	Word addressable mass storage
32	Prefers high speed mass storage

Word 14 — The assign mnemonic used on the @ASG or @CAT statement to create the file.

Word 17

inhibit-flags — These bits, when set, indicate inhibit options on @ASG control statements as follows:

Bit		
29	@ASG,G	Guarded file
28	@ASG,V	Inhibit unload
27	Not @ASG,P	Private file
26	@ASG,X	Exclusive use
25	@ASG,W	Write-only
24	@ASG,R	Read-only

Word 27

user-unit selection-indicators — These bits, when set, indicate:

Bit	
35	file created via device specification
34	file created via device specification
33	file created with logical placement (vs. absolute)
32	file is spread across devices
30-18	LDAT of initially selected device

Table 7-12. Mass Storage File Main Item - Sector 1

	35	32	29	23	17	11	0
0	U	0	0	0	link-to-sector-2-of-main-item-if-U = 0		
1	qualifier						
2							
3	filename						
4							
5	'*NO. 1*'						
6	link-to-sector-0-of-main-item						
7	nbr-of-backup-reels			nbr-of-two-word lapse-entries		absolute-F-cycle nbr-of-this-file	
8	date-and-time-of-backup-file-creation (TDATE$ format)						
9	unused		tape mode-codes	unused		total-nbr-of-1800-word-text-blocks	
10	tape-noise-constant			starting-file-position of-first-backup-reel		nbr-of-words written-in-last-block	
11	reel-nbr-of-first-reel-of-backup-file						
12	reel-nbr-of-second-reel-of-backup-file						
13	first-lapse-entry-or-all-zeros						
14							
15	second-lapse-enry-or-all-zeros						
16							

Table 7-12. Mass Storage File Main Item – Sector 1 (continued)

17	*unused*	*nbr-disk-pack-entries*
18 19	*first-disk-pack-entry*	
20 21	*second-disk-pack-entry*	
22 23	*third-disk-pack-entry*	
24 25	*fourth-disk-pack-entry*	
26 27	*fifth-disk-pack-entry*	

Word 9

tape-mode-codes — Indicates type of peripheral containing file. Same as equipment codes. These octal bits when set indicate:

002	Software translate
004	Hardware translate
010	Low density
020	Medium density
030	High density
040	Even parity

Words 13–14

first-lapse-entry or-all-zeros — Word 13: Time of first write following backup file creation
Word 14: Time of recovery when backup copy becomes primary copy
Date and time is given in TDATE$ format.

Word 18 – 19

first-disk-pack-entry — Word 18: Pack-id of the disk pack (six characters in Fieldata)
Word 19: Link to main item on disk pack

Table 7-13. Main Item - Sectors 2-n

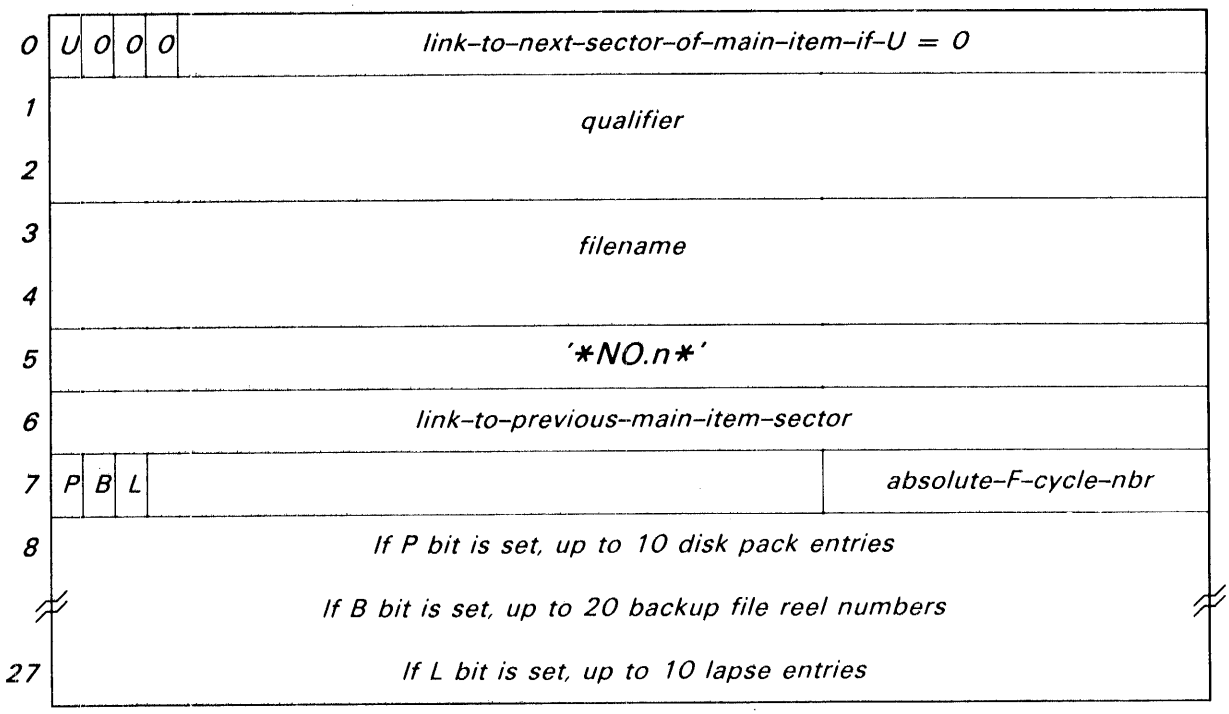


Table 7-14. Tape File Main Item - Sector 0

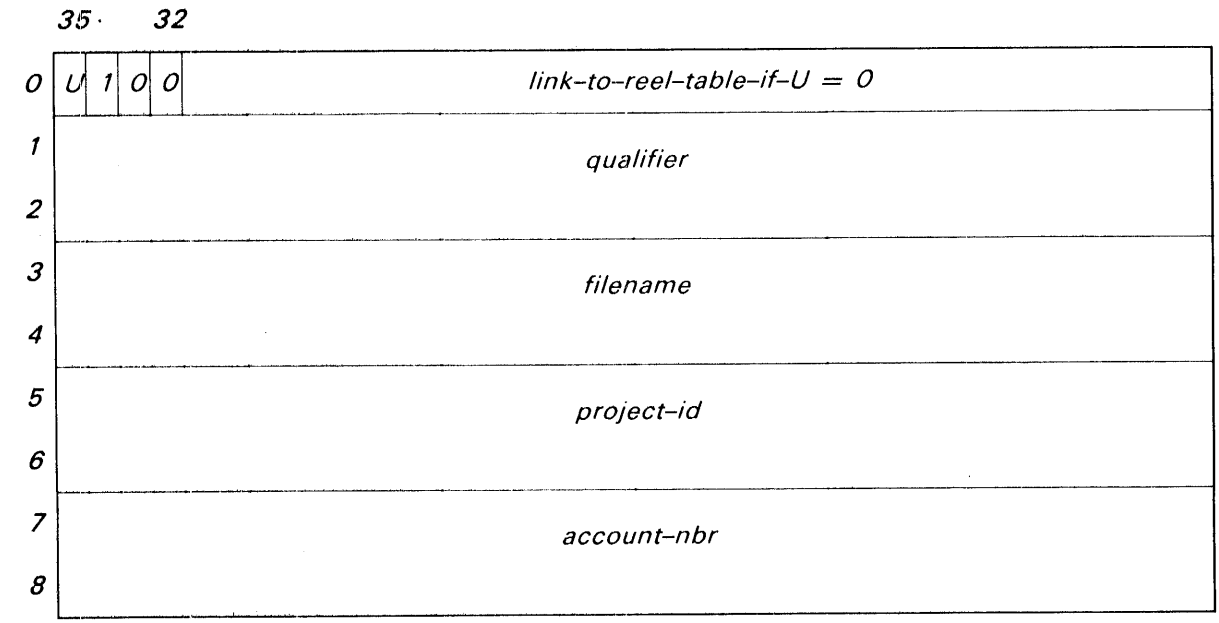
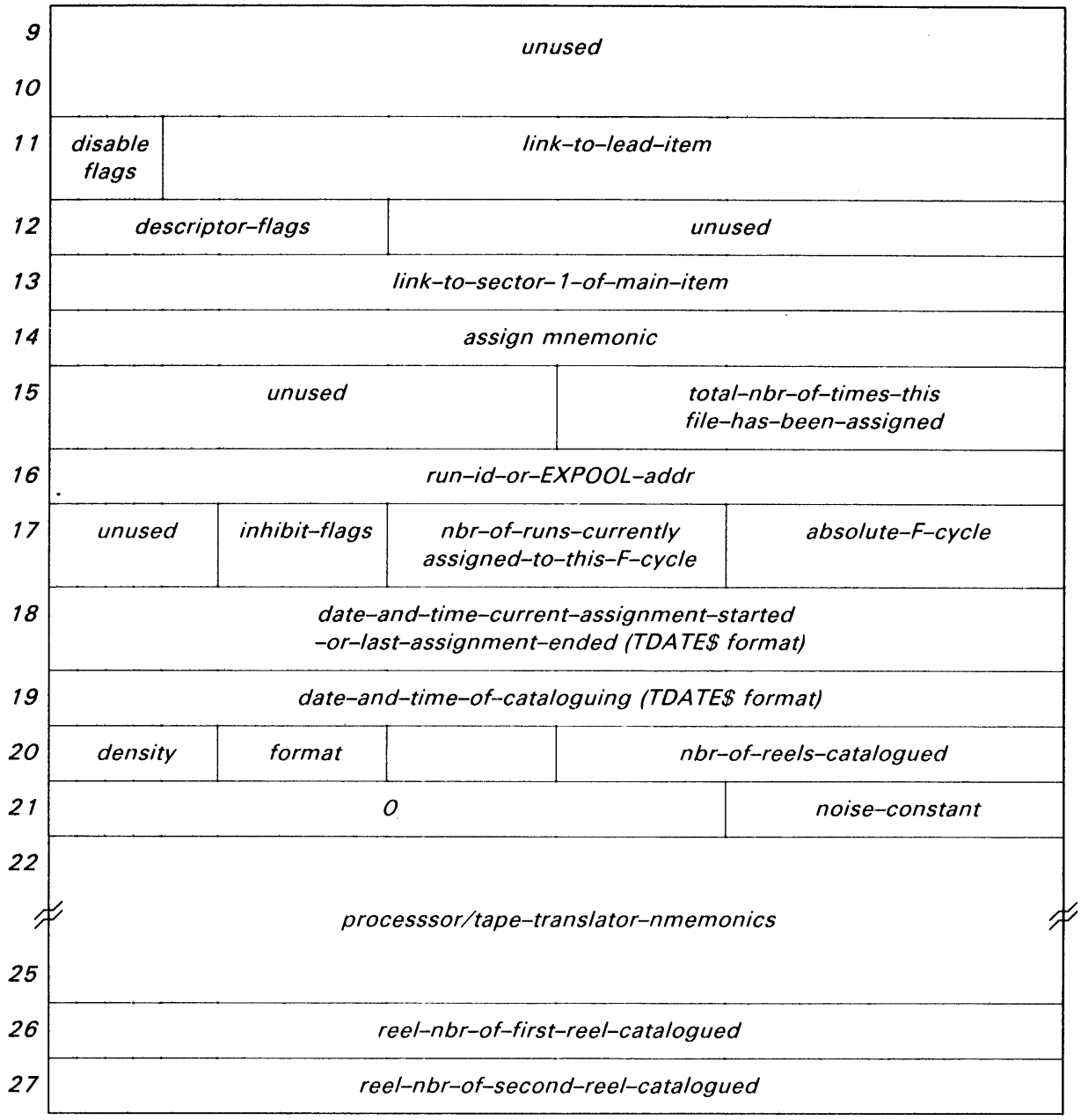


Table 7-14. Tape File Main Item - Sector 0 (continued)



Word 11

disable-flags, bits 35-32

1100_2	Facilities reject (file disabled because it has been destroyed)
1010_2	Facilities warning (file disabled because of an incomplete write)
1001_2	SECURE processor reject (file disabled because it was rejected by the SECURE processor).

Word 12

descriptor-flags

These bits, when set, indicate:

BIT	
35	Unused
34	Backed up
33	Backup file cannot be read
32	Unused
31	Bad main item sector 1
30	Old item format (created on EXEC Level 32 or earlier)
28-29	Tape file
27	Unused
26	File is to be made write-only
25	File is to be made read-only
24	File is to be dropped

Word 13

link-to-sector-1 of-main-item

Sector 1 of main item for tape files is identical in format to that for mass storage files except there are no disk pack entries.

Word 17

inhibit-flags

These bits, when set, indicate inhibit options on @ASG control statements as follows:

BIT		
29	ASG,G	Guarded file
28	Unused	
27	Not @ASG,P	Private file
26	Unused	
25	@ASG,W	Write only
24	@ASG,R	Read only

Word 20

density

For seven track tapes, the possible values and their meanings are:

01	200 FPI
02	556 FPI
03	800 FPI

For nine track tapes, the possible values and their meanings are:

01	800 FPI
02	1600 FPI
03	6250 FPI

format

The possible values and their meanings are:

001	Data converter
002	Quarter word

004 Six-bit packed
010 Eight-bit packed
020 Even parity
040 Set to nine track tape

Table 7-15. Removable Disk Pack Main Item (Sector 0)

	35	32	S2	S3	S4	T3
0	U	1	0	0	link-to-DAD-table-if-U = 0	
1	qualifier					
2						
3	filename					
4						
5	project-id					
6						
7	account-nbr					
8						
9	unused					
10	time-of-first-write-following-unload-or-backup-time (TDATE$ format)					
11	disable-flags		zero*			
12	descriptor-flags		block-buffering-EOF-sector-addr			
13	PCHAR-flags		link-to-section-1-of-main-item			
14	assign mnemonic					
15	unused		total-nbr-of-times this-file-has-been-assigned			
16	unused					
17	unused	inhibit flags	unused	absolute-F-cycle-nbr		
18	date-and-time-last-assignment-ended (TDATE$ format)					
19	date-and-time-of-cataloguing (TDATE$ format)					
20	initial-nbr-of-granules reserved-for-this-assignment		nbr-of-granules-of-quota-group-1			

Table 7-15. Removable Disk Pack Main Item (Sector 0) (continued)

21	*maximum-nbr-of-granules*	*nbr-of-granules-of-quota-group-2*
22	*highest-granule-nbr-assigned*	*nbr-of-granules-of-quota-group-3*
23	*highest-granule-nbr-written*	*nbr-of-granules-of-quota-group-4*
24	*three-most-significant characters-of-read-key**	*nbr-of-granules-of-quota-group-5*
25	*three-least-significant characters-of-read-key**	*nbr-of-granules-of-quota-group-6*
26	*three-most-significant characters-of-write-key**	*nbr-of-granules-of-quota-group-7*
27	*three-least-significant characters-of-write-key**	*nbr-of-granules-of-quota-group-8*

* The format pictured is applicable only to this MFD item as it appears on the removable disk pack.

Word 10

time-of-first-write following-unload-or-backup-time — If word 12, bit 35 = 1, then word 10 contains the unload time. If word 12, bit 35 = 0, then word 10 contains the time of the first write after the backup file was created or all zeros.

Word 11

disable-flags, Bits 35-32:

1100_2	Facilities reject (file disabled because it was destroyed)
1010_2	Facilities warning (file disabled because it was an incomplete write)
1001_2	SECURE processor reject (file disabled because it was rejected by the SECURE processor)

Bits 31-0: Zero (no link to the lead item is necessary since lead items are not contained on removable disk packs).

Word 12

descriptor-flags — These bits, when set, indicate:

Bit	
35	Unloaded
34	Backed up
33	Unused
32	Has lapse entries

31	Bad main item sector 1
30	Old item format (created on Executive Level 32 or earlier).
29	Unused
28	Communication between recovery and MFD maintenance routines for removable disk files. Extension sectors of the main item may have been changed on permanent storage. This assures the subsequent update of extension sectors on the disk packs.
27	Removable disk file
26	File is to be made write-only
25	File is to be made read-only
24	File is to be dropped

Word 13

PCHAR-flags — These bits, when set, indicate:

Bit	
35	Position granularity
34	Not used
33	Word addressable
32	Prefers high speed drum

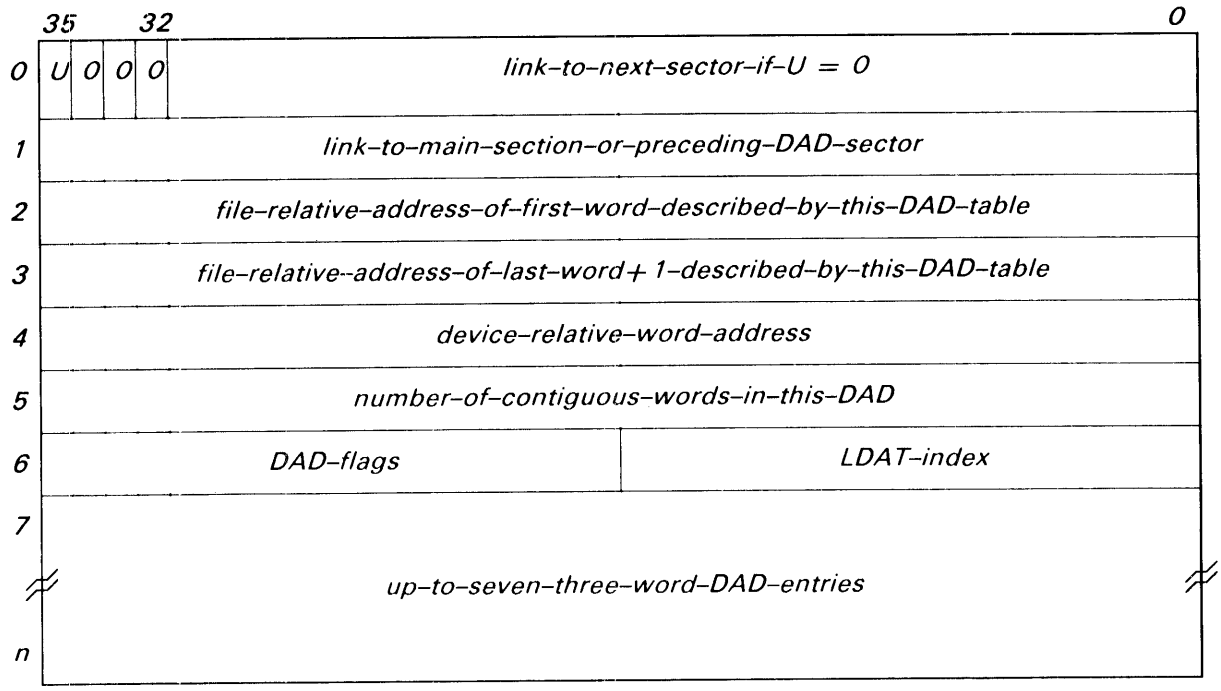
link-to-sector-1 of-main-item — Link to sector 1 of main item on removable disk.

Word 17

inhibit-flags — These bits, when set, indicate inhibit options on @ASG control statements as follows:

Bit		
29	@ASG,G	Guarded file
28	@ASG,V	Inhibit unload
27	Not @ASG,P	Private file
26	@ASG,X	Exclusive use
25	@ASG,W	Write-only
24	@ASG,R	Read-only

Table 7-16. Mass Storage File DAD Table



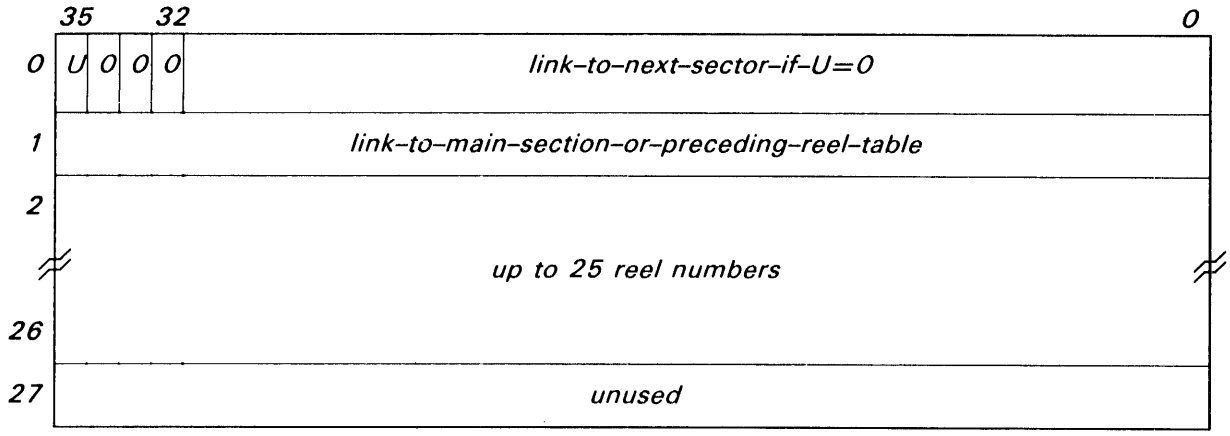
Word 1

When there is more than one DAD table, the second, third, and n^{th} DAD tables are each linked back to the preceding DAD table, and the first DAD table provides the link back to the main item.

Word 4

DAD Flags	01 – DAD has been badspotted 02 – Removable disk 04 – Last DAD in this DAD table
LDAT Index	Index into logical device address table

Table 7-17. Tape File Reel Table



If there are more than 25 reel numbers then there is more than one reel item. Each reel item lists from 1 to 25 reel numbers. When there is more than one reel item, the second, third, and n^{th} reel items are each linked back to the preceding reel item, and the first reel item provides the link back to the main item.

7.1.3. F-Cycles

A relative F-cycle number is associated with a particular file, relative to the Master File Directory lead item state at the time of assignment. An MFD lead item contains a link to each file and directory information that is common to all files in the set. Each link occupies one word in the MFD lead item starting in word 11 (see Table 7-8).

For more detailed information on F-cycle philosophy see SPERRY UNIVAC 1100 Series Executive System, Volume 2, EXEC Programmer Reference UP-4144.2 (current version).

The positive main item link which is nearest to or actually resides in word 11 can be referenced as relative F-cycle number 0. The positive main item link which is next nearest to word 11 can be referenced as relative F-cycle number -1, and so on through the remainder of the lead item.

Therefore, the relative F-cycle number associated with a particular file will change as other files are added to or deleted from the associated F-cycle set.

Figure 7-2 shows the relationship between absolute and relative F-cycle numbers for a particular F-cycle set. For this set, the maximum number of consecutive absolute F-cycles (S3 of word 9) has been set to 6. The current range of absolute F-cycle numbers (S4 of word 9) is 5 and the highest absolute F-cycle number (T3 of word 9) is 5. For the complete lead item format see Table 7-8).

8						
9		04	06	05	0005	
10	0					
11	0	link to absolute F-cycle 5				Link to relative F-cycle 0
12	0	link to absolute F-cycle 4				Link to relative F-cycle -1
13	0	link to absolute F-cycle 3				Link to relative F-cycle -2
14	0	0				
15	0	link to absolute F-cycle 1				Link to relative F-cycle -3
16	0	0				
17	0	0				
18	0	0				
19						

Figure 7-2. Relative Absolute F-Cycle Relationships Example 1

When a new file is being generated, the link is inserted into the proper location in the lead item and the other links are moved down if necessary. The counts in word 9 are adjusted accordingly. If the file is being created by an @ASG,C or @ASG,U the link will have bit 35 set.

The file becomes catalogued (bit 35 in the link is cleared) when the file is freed or at run termination. When bit 35 is set in the link from the MFD lead item, that file is not considered in the relative cycle numbering scheme and therefore it is not included in S2 of word 9 (number of catalogued F-cycles in the lead item). For example, if one adds two files to the lead item shown above by @ASG,C FN(7) and a @ASG,C FN(+1) the lead item looks as shown in Figure 7-3.

This sequence has pushed absolute F-cycle 1 out of the maximum allowable F-cycle range (S3 of word 9). Therefore the system deleted absolute F-cycle 1. If F-cycle 1 had been currently assigned, the system would have been unable to delete it, so the @ASG,C FN(7) would have received an F-cycle conflict (FAC REJECTED with bit 5 set in the status word).

The fact that F-cycle +1 is currently assigned prevents further cataloguing within this cycle set. Bit 34 in word 10 indicates this situation exists. When the +1 file is freed it will become relative F-cycle 0 and further cataloguing will be allowed. An attempt to catalogue a new file within the set when +1 file is being created will cause the F-cycle conflict FAC REJECTED, i.e., if the preceding example had specified the @ASG,C FN(+1) and then a @ASG,C FN(7), the @ASG,C FN(7) would be rejected.

There is one other restriction on F-cycle generation that will cause an F-cycle conflict FAC REJECTED. For a new cycle to be created, its absolute F-cycle number must be within the following range:

$$(x-w)<z<=(x-y+w+1)$$

where:

z absolute F-cycle number requested

w S3 of word 9 of the lead item (maximum number of F-cycles)

x T3 of word 9 of the lead item (cycle number of latest F-cycle)

y S4 of word 9 of the lead item (current range of F-cycles)

This range prevents more than one file from being dropped from the set (the oldest cycle) when a new file is being added to the set.

8			*w*	*y*	*x*	
9		03	06	06	0010	
10	2					
11	4	link to absolute F-cycle 010				Link to relative F-cycle +1
12	4	link to absolute F-cycle 7				
13	0	0				
14	0	link to absolute F-cycle 5				Link to relative F-cycle 0
15	0	link to absolute F-cycle 4				Link to relative F-cycle -1
16	0	link to absolute F-cycle 3				Link to relative F-cycle -2
17	0	0				
18	0	0				
19						

Figure 7-3. Relative Absolute F-Cycle Relationships Example 2

7.1.4. Master File Directory Manipulation (MSCON$)

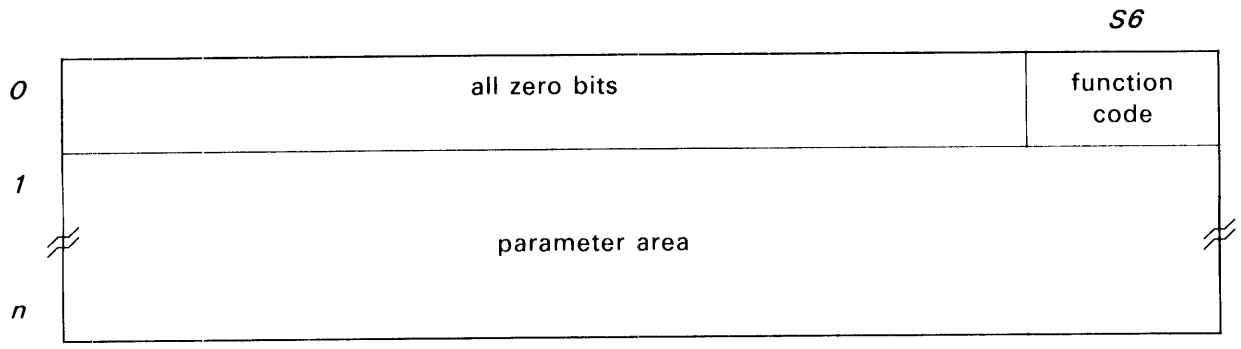
The MSCON$ Executive Request (ER) enables the user to obtain either the entire MFD or entries pertaining to a particular file. In addition, MSCON$ provides the means of altering indicators in the directory items.

Several of the MSCON$ functions can be performed only by privileged runs. Classified file information is revealed only to privileged runs. A privileged run is one that has the SYS$*DLOC$ file assigned to the run with the correct read/write keys. The SYS$*DLOC$ file is a system file which is created at system initialization. Its primary use is to identify privileged runs. At the time a privileged run issues a @FREE of the file SYS$*DLOC$, the privileged mode of operation is terminated.

The general format of the MSCON$ request is:

```
L,U  A0,pktaddr
ER   MSCON$
```

The packet address loaded into register A0 references a packet having the following general format:



The function code in S6 of word 0 indicates the particular control routine desired. The parameter area is a communications area used to pass information from the user to the ER. In cases where words 1 and 2 specify an internal filename, that file must be assigned to the calling run.

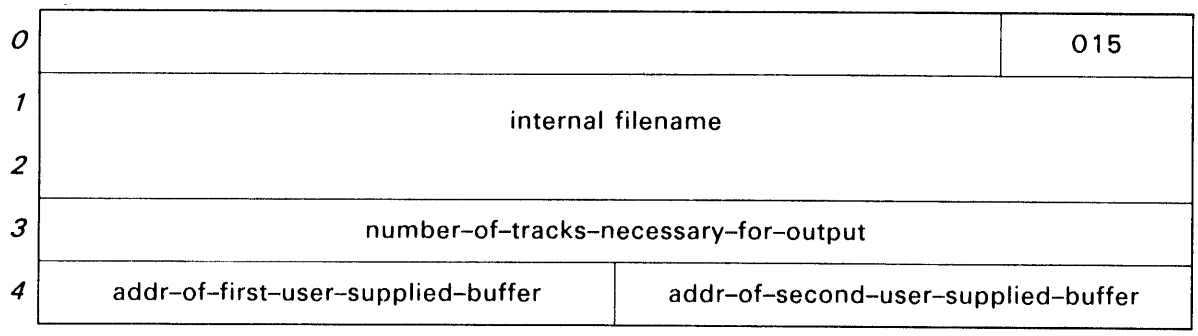
The octal function codes are as follows:

0015 - DGET$
0016 - DGETP$
0020 - DREAD$
0030 - DBITS$
0031 - DBACK$
0032 - DLAPS$
0033 - DUNLD$
0034 - DCYC$
0035 - DKEY$
0036 - DBB$
0037 - DREG$
0060 - MSALL$
0160 - MSSUM$

7.1.4.1. Item Retrieval For All Files (DGET$)

This function creates a file containing the directory items. This file effectively acts as a checkpoint of the MFD.

The format of the packet in the MSCON$ request is:



Words 1 and 2

internal filename — Specifies the internal filename of the output file to be created.

Word 3

number-of-tracks-necessary-for-output — Specifies the number of tracks necessary for the DGET$ output file.

Word 4

addr-of-first-user specified-buffer — Specifies starting address of a track-size (1792 words), user-supplied buffer.

addr-of-second-user specified-buffer — Specifies starting address of a track-size (1792 words), user-supplied buffer.

If the output file's initial reserve is sufficient in length to accomplish the DGET$ request, word 4 is used to set up double-buffered I/O for transferring MFD information to the output file. If, however, the output file's initial reserve is insufficient to contain the current total of directory items, a value is returned in word 3 that represents the number of tracks needed to perform the DGET$ request. This request is rejected and the appropriate error status is returned in register A0.

Because of the sensitive nature of the information to be contained in the buffers, this function will inspect the user PCT to determine if:

- the user has any I/O request outstanding
- the user is employing ESI activity
- the user's current activity count is more than one
- the user is attached to any common bank and either of the buffers supplied lies in a common bank

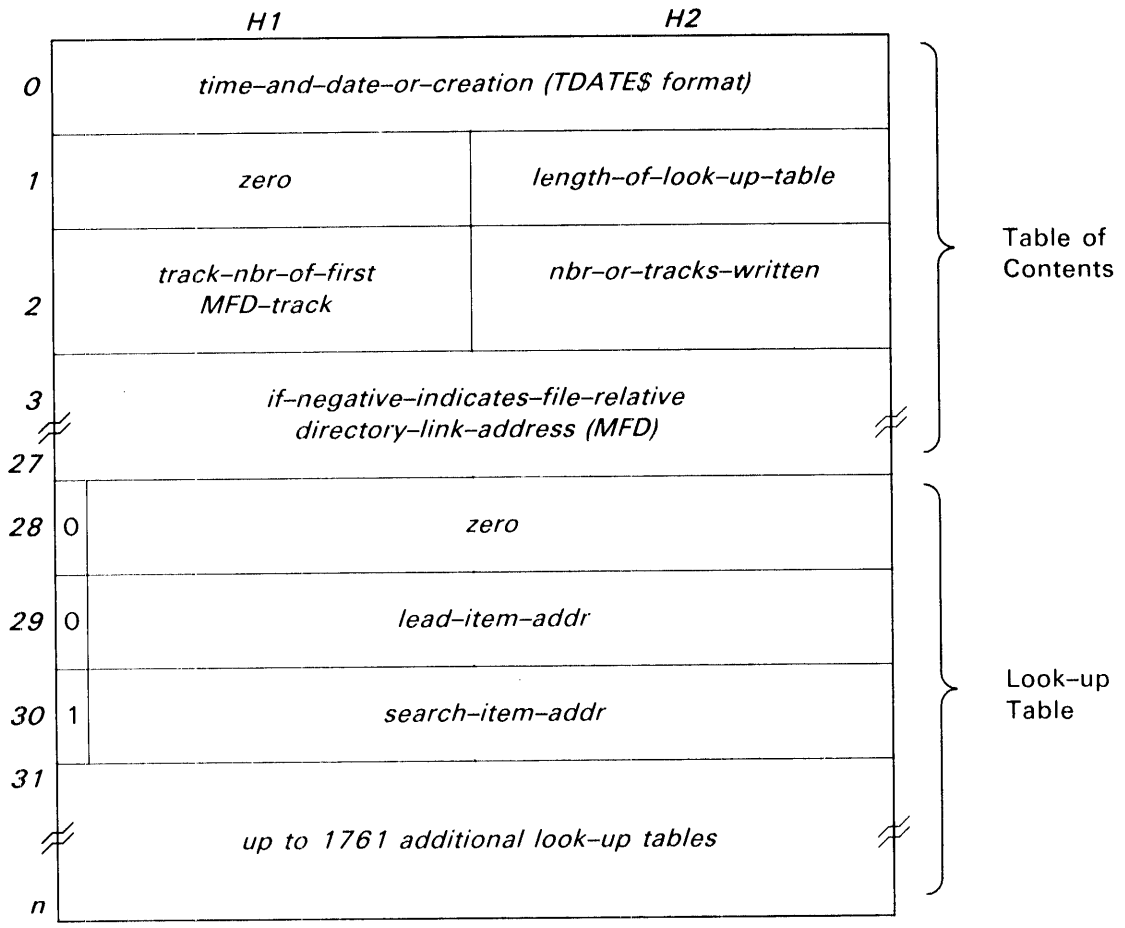
If any one of the above conditions exists, this request is rejected and the appropriate error status is returned in register A0.

If the calling run is not privileged, the read/write keys are set to blanks and the project-id and account number, if different from those of the run, are set to //////.

If the calling run is privileged, only files catalogued with a G option have the read/write keys and the project-id altered in this way.

The output from the DGET$ function consists of a table of contents (starting at sector 0), look-up table (starting at sector 1) and directory items (lead, main, granule, and search items starting at sector 0 of the track following the look-up table). The output is written into the user file named in the input packet. The output track formats are:

- Track 0



NOTE:

Words 0 through 27 are the Table of Contents, words 28 through n are the Look-up Table.

Word 0

The date and time when the DGET$ function created this file, where:

S1 - Month
S2 - Day
S3 - Year (modulo 1964)
H2 - Time in seconds from midnight

Word 1

length-of look-up-table — This table starts in sector 1 of the first track. If the look-up table is greater than 63 sectors, the remainder is written in successive tracks.

Word 2

track-nbr-of first-MFD-track — If the look-up table is located in the first two tracks, the number in this word is 3.

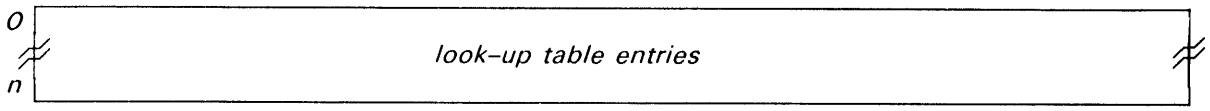
number-of tracks-written — The number of tracks written by the DGET$ function in the user's file.

Word 28

This word is the start of the look-up table. (The length of the look-up table is equal to the DCLUTS tag in the Executive. This value is supplied by the DGET$ function in H2, word 1 of the output file). The first 63 sectors (1764 decimal words) of the look-up table are written in track 0. If there are more than 1764 possible entries in the look-up table, the remaining entries are written in the following tracks as needed.

The formats of look-up table entries are illustrated in words 28 through 30; the entry is zero if no file has a look-up index equal to the word number. This type of format is shown in word 28. If only one file has this index, the file's lead item address is stored in this word, as shown in word 29. If more than one file has this index, the entry contains the address of a search item, as shown in word 30.

- Track 1, where the length of the look-up table exceeds 1764 decimal words.



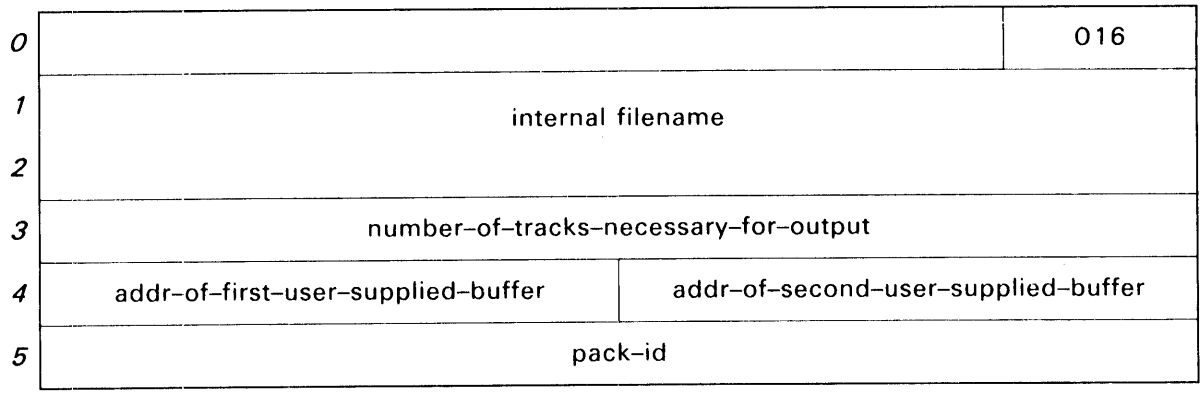
- Track (n//28+1)//64, where n = length in words of look-up table.

  Start of the MFD tracks. The first MFD track written for each unit contains a Directory Allocation Sector (DAS) as sector 0 of that track. Tracks for each unit are written to the user's file in the same order as listed in the allocation sector. The format of the DAS is shown in Table 7-7.

- Tracks (n//28+1)//64+1, where n=length in words of the look-up table, through n-1). The value of n is found in the table of contents (track 0, sector 0) in H2 of word 2.

7.1.4.2. Item Retrieval For Disk Packs (DGETP$)

This function creates a file in a similar manner as DGET$ (7.1.5.1). The only difference is that the directory items from a specified pack are inserted into the output file (the pack must be mounted and assigned). DGETP$ imposes the same security constraints as detailed for DGET$. Sector 0 of the output file for DGETP$ differs from that for DGET$ in that it contains the equipment subcode in word 3, S6. The format of the packet used in the MSCON$ request is shown below.



Words 1 through 4

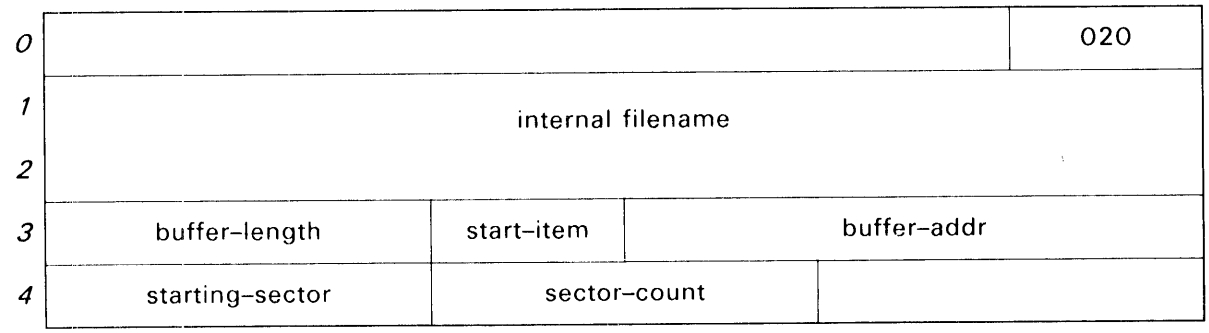
Same as for DGET$ function (see 7.1.5.1).

Word 5

A 1- to 6-character alphanumeric that specifies the pack-id written on the disk pack.

7.1.4.3. Item Retrieval For an Individual File (DREAD$)

This function provides a full complement of MFD information for a single F-cycle of a file set. The format of the packet used in the MSCON$ request is:



Words 1 and 2

internal filename — Specifies the internal filename of the file for which the MFD information is to be retrieved.

Word 3

buffer-length — Length (in words) of the buffer into which the directory items are to be written.

start-item — Specifies the MFD items desired:

- 00 Start with the lead item, and then continue with the main item and DAD tables (in ascending order) until the end of the MFD for this file or the buffer limit is reached.
- 01 Same as above, but start with main item.
- 02 Same as above, but start with DAD table.

buffer-addr — Address of buffer into which the directory items are to be written.

Word 4

starting-sector — Sector number within the starting item at which retrieval is to start (for example, start at sector 26 of DAD table).

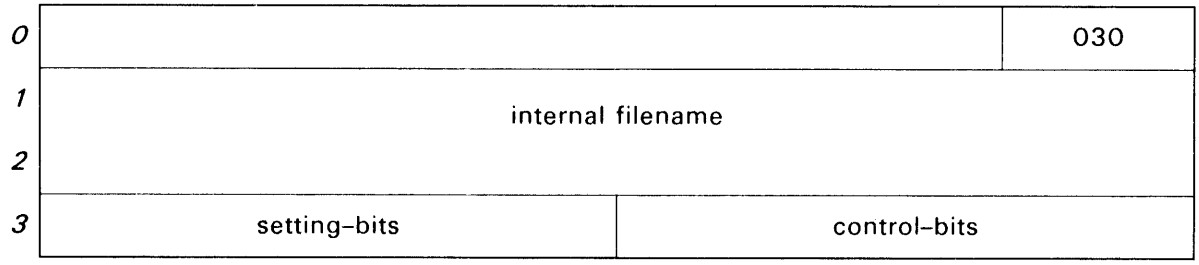
sector-count — Number of sectors written in user buffer (supplied by DREAD$ function).

The MFD information supplied by the DREAD$ function appears in the output buffer in the order specified in the user packet. If the calling run is privileged and the subject file is not a G-option file, then any read/write keys and project-ids are passed unchanged. If the calling run is not privileged, or the subject file is a G-option file, then any project-id different from the user project-id is changed to slashes. The read/write keys are also changed except when the corresponding Program Control Table (PCT) read-key-correct and write-key-correct bits are set or the read/write keys are blank.

7.1.4.4. Altering Main Item (DBITS$)

This function allows a privileged user, or one who has assigned the subject file with the correct read/write keys, to change the value of certain fields in the main item, sector zero.

The format of the packet used in the MSCON$ request is:



Words 1 and 2

Specify the file wherein fields are to be changed.

Word 3

If bit 0 is set, use the contents of bit 18 for the disabled status.*
If bit 1 is set, use the contents of bit 19 for the hardware disabled reject (0400) status.*
If bit 2 is set, use the contents of bit 20 for the SECURE disabled reject (0100) status.*
If bit 3 is set, use the contents of bit 21 for the software disabled warning (0200) status.*
If bit 6 is set, use the contents of bit 24 for the to-be-made-write-only flag (W option).
If bit 7 is set, use the contents of bit 25 for the to-be-made-read-only flag (R option).
If bit 9 is set, use the contents of bit 27 for the public/private status (P option).
If bit 10 is set, use the contents of bit 28 for the unload inhibit flag (V option).

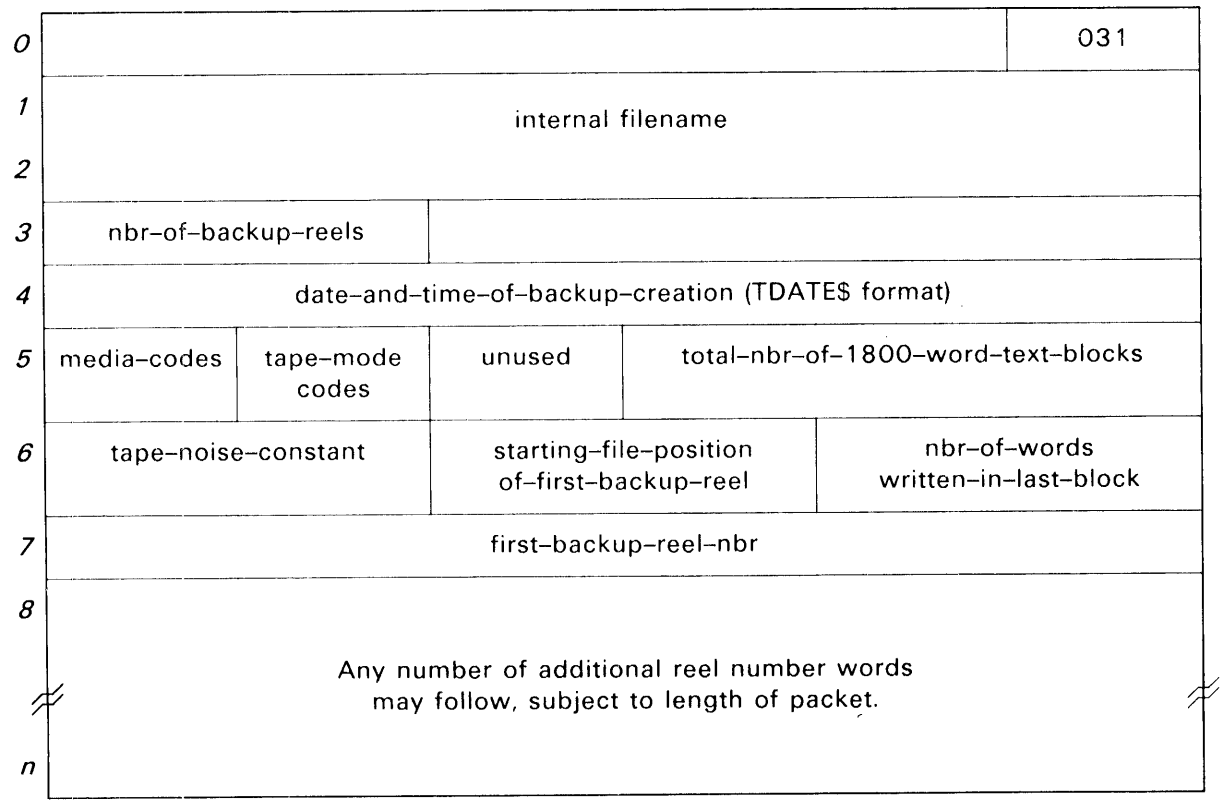
** NOTE:*

If any of bits 34, 33 or 32 in main item sector 0, word 11 are set, then bit 35 will be set. Conversely, if bit 35 is cleared, then bits 34, 33, and 32 will be cleared.

7.1.4.5. Altering Backup File Entries (DBACK$)

This function allows the user to change certain fields pertaining to backup files. In processing this request the DBACK$ function clears word 10 (time stamp of first write after backup) of main item sector 0, sets the backed-up bit in the main item descriptor (word 12), and clears the file-changed bit in PCFIDL of the PCT facilities item.

The format of the packet used in the MSCON$ request is:



Words 1 and 2

These specify the file entry being changed.

Words 3 – n

These words replace equivalent words in words 1–n of the main item.

Packet length = 7 + nbr of backup reels.

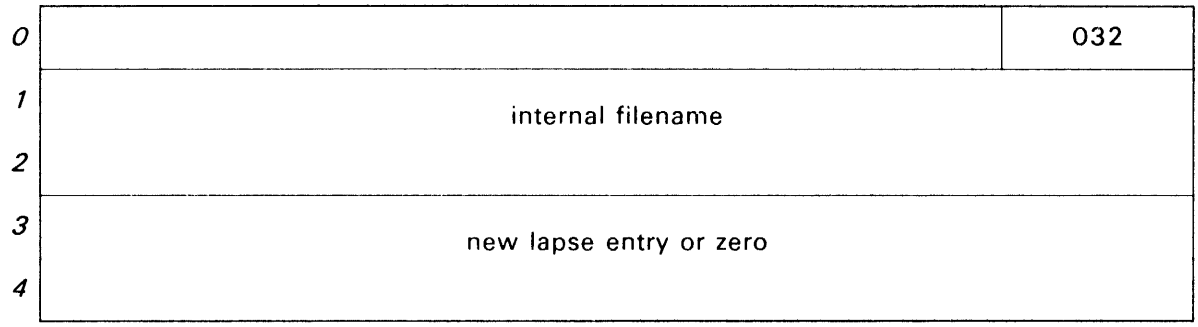
7.1.4.6. Altering Lapse Entries (DLAPS$)

This function allows the user to enter new lapse entries or to alter information pertaining to lapse entries.

If words 3 and 4 of the input packet are both nonzero, the DLAPS$ function enters the new lapse entry in the next available location, adds 1 to the lapse entry count in sector 1 of the main item, and sets the has–lapse–entries bit in the main item descriptor.

If word 3 of the input packet is zero, the DLAPS$ function zeros out the lapse entry count and clears the has–lapse–entries bit in the main item descriptor.

The format of the packet used in the MSCON$ request is:



Words 1 and 2

These specifies the file entry being changed.

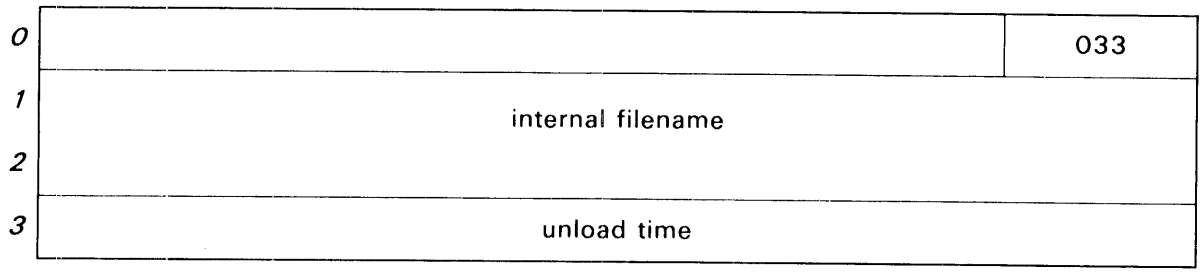
Words 3 and 4

The lapse entry.

7.1.4.7. Changing Unload Time (DUNLD$)

This function stores the unload time (nonzero for unload, zero for load) in sector 0 of the main item (word 10). When the unload time is nonzero, the DUNLD$ function ensures that the unload bit is set. When the unload time is zero, the DUNLD$ function ensures that the unload bit is cleared and that the this–file–has–been–changed bit in the PCT is cleared. This function is primarily for use by the SECURE processor.

The format of the packet used in the MSCON$ request is:



7.1.4.8. Changing Maximum Cycle Range (DCYC$)

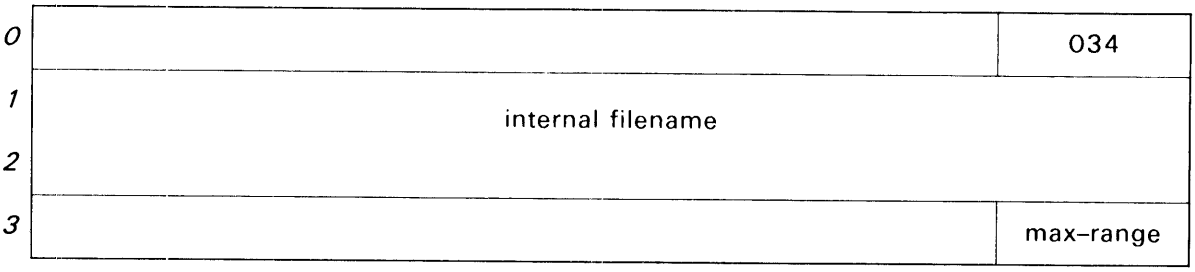
This function changes the "maximum-range" field in sector 0 of the lead item (S3 of word 9), if the specified maximum range value satisfies these criteria:

1. The specified maximum range is in the range 1 through 32_{10}.

2. The specified range is equal to or greater than the current-range value (S4 of word 9).

If these criteria are not satisfied, this request is rejected and the appropriate error status is returned in register A0. To meet the second restriction, it is necessary to first delete any F-cycles which would otherwise fall outside of the proposed maximum range.

The format of the packet used in the MSCON$ request is:



7.1.4.9. Changing Read/Write Keys (DKEY$)

This function allows the user to change file read and write keys. To do this, the old read and write keys must be furnished in the packet along with either or both of the new keys. If either key is not to be changed, a word of slashes (word 5 or 6 of the packet) is used to indicate this. To remove a key altogether, blanks are used.

If the old keys do not match the current keys, a check is made to see if the new keys match the current keys. If this is the case, DKEY$ will exit normally. If neither the old keys nor the new keys match the corresponding file keys, the DKEY$ request is rejected and the user run is aborted. If the file resides on removable disk packs, any one DKEY$ request will process key changes only for the F-cycle associated with the internal filename specified in the packet.

To accomplish the desired key changes for removable disk files with n F-cycles, it is necessary that the user either employ the FURPUR processor @CHG control statement, or use the following sequence n times:

```
@ASG,A   FN(F-cycle specification)        (Do not specify R/W keys.)
           •
           •
           •
   L,U   A0,DKEY$PACKET                   (DKEY$ PACKET must identify this F-cycle and both
   ER MSCON$                               old and new keys.)

@FREE    FN                               (F-cycle specification.)
```

In the event of system or run failure, the above sequence can be resubmitted without modification to ensure key changes for all n F-cycles.

The format of the packet used in the MSCON$ request is:

0		035
1	internal filename	
2		
3	old-read-key (blanks if none)	
4	old-write-key (blanks if none)	
5	new-read-key-or-blanks (slashes if no change)	
6	new-write-key-or-blanks (slashes if no change)	

7.1.4.10. Modifying File Identifier (DREG$)

This function was designed for use by the SECURE processor. It allows a privileged requestor to change certain fields in the lead and main items of catalogued files.

The time of cataloguing (word 8 of the caller packet) is stored into main item word 19. If the time of last reference (word 9) equals a nonzero value, it is stored into word 18 of the main item. If the account-number field (word 5 and 6) contains anything other than binary zeros or Fieldata blanks (05), it is used to overlay the main item account number. If the project-id (words 3 and 4) contains anything other than binary zeros, it replaces the project-id in both the lead and main items. If the u-s-f control bit (word 7, bit 18) is set, the file is a unit specified file and the u-s-f update bits (word 7, bits 23-19) are used to overlay the user unit selection indicators of the main item (word 27, bits 35-31). If the nbr-of-times-assigned field (word 7, H2) contains anything other than binary zeros, it replaces the total nbr-of-times-the-file-has-been-assigned field in the main item.

The format of the packet used in MSCON$ request is:

0			037
1–2	filename		
3–4	project-id		
5–6	account-nbr		
7	0	u-s-f	nbr-of-times-assigned
8	date-and-time-of-cataloguing (TDATE$ FORMAT)		
9	date-and-time-of-last-reference (TDATE$ FORMAT)		

7.1.4.11. Monitoring Mass Storage Availability (MSALL$ and MSSUM$)

This function enables a user program to monitor the availability and status of mass storage by transferring specific information from the equipment summary and unit status tables into a user-supplied buffer.

Upon successful completion of this function, H1 of packet word 2 contains the total number of image words transferred.

The format of the packet used in the MSCON$ request is:

0	revision indicator		function code
1	buffer length	buffer address	
2	length of equipment index table equipment summary, UST output		
3	first-mass-storage-type-(MSBEG)	last-mass-storage-type-(MSEND)	
4	first-disk-storage-type-(DSCBEG)	last-disk-storage-type-(DSCEND)	

Word 0

revision indicator — initially set to 1, this will be updated each time the table format is revised

function

060 (MSALL$) – information from equipment summary and unit status tables transferred

0160 (MSSUM$) – information from only equipment summary tables is transferred

The output consists of the mass storage portion of the equipment index table, then the selected information from the equipment summary and unit status tables for each mass storage type. The link addresses displayed are relative to the start of the user buffer. Any cells in tables not maintained for an equipment type or function (i.e., links to unit tables on MSSUM$ function) will be 0.

The table formats are as follows:

Equipment Index Table

0		number of entries
1	relative address of first unit table equipment type 1 (MSBEG)	equipment summary table relative address equipment type 1 (MSBEG)
2	relative address of first unit table equipment type 2 (MSBEG+1)	equipment summary table relative address equipment type 2 (MSBEG+1)
n	relative address of first unit table equipment type n (MSBEG+n)	equipment summary table relative address equipment type n (MSBEG+n)

Equipment Summary Table

0	number of units with this equipment index	number of units up or reserved with this equipment index	number of units up and not assigned
1	relative address of first unit table for this equipment index	relative address of unit table for this equipment to be selected next	
2	specific equipment mnemonic		
3	number of tracks available this equipment		

Word 1

next-to-be-selected unit table

unit next-to-be-selected for allocation based on a facilities algorithm

Mass Storage Unit Table

0	LDAT index	relative address of next unit table for same equipment type	
1	device mnemonic name		
2	unit status bits		
3	unit status bits		
4	run-id of PCT to which unit is allocated		
5	equipment code index	assign count	
6	number of references since last error		
7	media-id (pack for disk)		
010	number of tracks available		
011	heads per cylinder	words per record	records per track

Word 2

Bit	0	unit received unsolicited interrupt
	1	unused
	2	unit is down
	3	unit is reserved
	4	unit is suspended
	5	unit is not available
	6	hold on subsequent I/O to unit
	7	impasse – message outstanding

	8	impasse – MV keyin in progress
	9	impasse – label check necessary
	10	impasse – multiple attention interrupt
	11	impasse – label checking errored
	12	impasse – unit not available

Word 3 status bits set to 1 have following meaning:

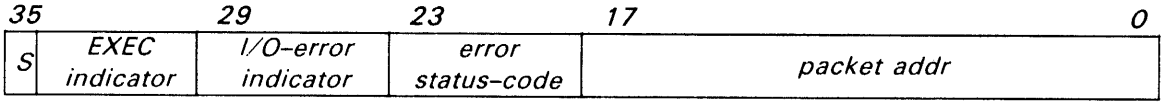
Bit	0	unused
	1	fixed storage
	2	unused
	3	unit is saturated
	4	allocation in progress
	5	unit in state of flux
	6	label check required
	7	further assignments on this unit to be aborted
	8	to be initialized
	9	MFDF DAS allocation in progress

Word 7

media–id	If disk equipment and zero, no pack is mounted or requested for this unit. Pack–id is EXEC 8 if prepping is in progress
Word 011	Zero for non–disc equipment

7.1.5. MSCON$ Status Conditions

When MSCON$ returns control to the user, register A0 contains the original packet address in H2, possible error status codes in bits 29 – 18 and an EXEC–indicator in bits 34 – 30.



The value of the EXEC–indicator is supplied by MSCON$ for use by the SECURE and FURPUR processors.

If bit 35 = 0 and bits 29 – 18 = 0, this signifies normal completion of the requested function.

If bit 35 = 0 and bits 23 – 18 contain 01, the 01 is a special status code returned by the DREAD$ function, signifying that the end of the user buffer has been encountered and there are more directory items to be returned.

If bit 35 = 1 and the error status code is 024, the value in bits 29 – 24 is the I/O status code received by MSCON$.

If bit 35 = 1 and the error status code is other then 024, the possible status codes are:

020	Wrong MSCON$ function code in user packet.

021 User packet not within program limits.

022 Referenced file is not assigned to this user.

023 User is referencing a temporary file.

025 User buffer not within program limits.

026 User is referencing a nonexistent start item (returned by the DREAD$ function, see 7.1.4.3).

027 User buffer area not large enough (returned by MSALL$ functions, see 7.1.4.11). Or, user packet specifies zero for number of backup tape reels (returned by the DBACK$ function, see 7.1.4.5).

030 The function requires a main item extension sector which does not exist for this file (returned by DLAPS$ function, see 7.1.4.6).

031 The referenced disk unit has been marked down or reserved (returned by the DGETP$ function, see 7.1.4.2).

032 The user packet specifies an illegal pack-id, or the requested pack-id cannot be found in the unit status tables, or the specified pack is marked for label checking but label checking cannot be completed, or the specified pack is being prepped, or the pack is not assigned to the user (returned by the DGETP$ function, see 7.1.4.2).

033 The output file initial reserve is too small to contain the current total of system directory items (returned by the DGET$ function, see 7.1.4.1).

034 The cumulative total of system directory items has dynamically expanded beyond the capacity of the output file (returned by the DGET$ function, see 7.1.4.1). This situation differs from that described for status code 033, in that, in this instance DGET$ has been in process and directory items have been placed on output to the file.

035 The user program has I/O outstanding, is employing ESI activity, has a count of activities totaling more than one, or is supplying a data buffer which lies in a common bank (returned by functions DGET$, see 7.1.4.1 and DGETP$, see 7.1.5.2).

036 The packet specifies a maximum range value not in the range $1\text{-}32_{10}$, or less than the current range value (returned by the DCYC$ function; see 7.1.4.8).

037 The user is neither privileged, nor has the subject file been assigned with correct read/write keys (returned by the DBIT$ function, see 7.1.4.4).

7.1.6. Down By Track (ER BDSPT$)

ER BDSPT$ provides a means for the user to remove a track from the system's mass storage pool. Input to BDSPT$ can be a file relative granule or a list of Device Area Descriptors.

ER BDSPT$ can only be performed by privileged runs (see SPERRY UNIVAC 1100 Series Executive, Volume 2, EXEC Programmer Reference UP-4144.2 (current version).

7.2. FILE PRESERVATION AND RECOVERY

7.2.1. SECURE

One function of the SECURE processor is to protect the physical security of catalogued files, whether they reside on mass storage or removable disk, by creating backup copies on tape. These tapes are an integral part of the sophisticated catalogued file recovery process available through SECURE.

It should be noted that any SECURE operation described may handle the complete catalogued file set, or be limited to various specific file subsets.

For more detailed information on SECURE see SPERRY UNIVAC 1100 Series Executive System Volume 3, System Processors Programmer Reference, UP-4144.3 (current version).

7.2.1.1. SAVE

If a new backup is to be created for a file because a write operation has been done to it since the last backup was created, a SAVE operation should be performed. The various tapes created on SAVE operations may be consolidated into a new set of backups by the SAVE ALL command, which produces a new backup for a file regardless of whether the present backup is outdated or not.

The availability of up-to-date backup copies of files is essential if the site is to operate with a data base whose size is larger than its mass storage capabilities. They are also valuable for text recovery in case the mass storage copy of a file becomes destroyed.

7.2.1.2. UNLOAD

The UNLOAD operation is to be done when it is desirable to increase the amount of mass storage available for new allocation. In addition, the unload capability (when called by the REVERT command) is useful when it is necessary to replace a file's mass storage copy of the text with its backup copy. After a REVERT is done, a subsequent assign of the file causes the text of the file to be loaded from the backup tape. On the UNLOAD statement, onsite personnel may specify the fileset whose granules are to be released, or the amount of tracks to be released. If the amount of tracks is specified, SECURE uses the internal algorithm called the UEF to select the files most eligible for unloading.

The UEF computation may be modified to enable the weighting factors to be influenced by source statements in the rollout jobstream. The present formula is as follows:

$$UEF = (CURNCY * A + FREQCY * B + SIZEWT * C)/32 + D + E + F + G$$

where:

CURNCY — is the currency weighting factor with a default value of 30 which may be altered by the source statement CURRENCY = *nn*

A — is a currency code in the range 0 to 32 and is a measure of the files currency as indicated by the time of last reference. The code is extracted from UEFASG, a table of time periods defined in the element SETUP.

FREQCY — is the frequency weighting factor with a default value of 20 which may be altered by the source statement FREQUENCY = *nn*

B — is a frequency code in the range 0 to 32 and is a measure of the average time between references. The code is extracted from the same table as the currency code.

SIZEWT — is the size weighting factor with a default value of 10 which may be altered by the source statement SIZE = *nn*

C — is a size code in the range 0 to 32 based on the file size. The code is extracted from the table UEFTRK defined in the element SETUP.

D — is a bias value based on equipment type and is added if the file resides on high speed drum. The default value is 2 which may be altered by the source statement EQUIPMENT = *nn*

E — is a bias value based on whether the file is catalogued as public or private and is added if the file is private. The default value is 1 which may be altered by the source statement PRIVATE BIAS = *nn*

F — is a bias value based on whether the file has associated with it any F-cycles which are more recent than the one being examined. The default value is 1 which may be altered by the source statement FCYCLE = *nn*

G — is set by the source statement UEF = *nn* and is used to selectively add or subtract a number *nn* from the computed UEF.

7.2.1.3. DUMP

The DUMP statement is provided in the SECURE source language to provide the programmer with the ability to display certain internal tables and to patch the absolute program for temporary resolution of bugs. It has the following formats:

```
DUMP BACKUP
```

This format indicates that the backup tables are to be dumped.

```
DUMP CARDIM
```

This format indicates that the CARDIM table, which contains the internal format of the source language statements, is to be dumped.

```
DUMP SURVEY
```

This format indicates that all the survey items are to be dumped.

DUMP *addr, fix1, fix2, ...*
or
DUMP *n, addr, fix1, fix2, ...*

These formats indicate that the SECURE processor is to be patched. The value of *fix1* is stored at *addr*, the value of *fix2* is stored at *addr* + 1, and so on. The specifications after *fix1* are optional. All values may be decimal or octal; octal values must be indicated by a leading zero. The value of *n* determines the time at which the patch is performed, as indicated in Table 7-18.

Table 7-18. DUMP Patch Timing Indicators

n	Patch Time
omitted	Immediate (i.e., during source language processing).
0	When SETUP3 segment is loaded.
1	When SORT4 segment is loaded.
2	When ACTN5 or RASTR5 segment is loaded.
3	When LIST6 segment is loaded.

7.2.1.4. REMOVE

REMOVE operations are used to decatalogue specific files or sets of files from the Master File Directory. This is especially useful when a site's data base contains a number of files being referenced very infrequently. When a REMOVE is done in conjunction with SECURE's file archiving mechanism, much overhead of directory and text manipulation for these 'dormant' files is relieved. However, SECURE can readily recover these files when necessary. When a removable disk pack becomes inaccessible either due to equipment failure or planned departure from the site, the files catalogued on the disk pack must be REMOVE'd. In this example, only the mass storage copy of the directory for these files is destroyed, the directory on the disk pack itself remains intact.

7.2.1.5. REGISTER

To catalogue files from either SECURE-created backup tapes, a Master File Directory snapshot tape, or removable disk packs, a REGISTER operation is needed. After an initial boot, a REGISTER DIRECTORY TAPE FROM IBACKUP operation (with IBACKUP equated to the last MFD snapshot tape created) will recatalogue all files catalogued previous to the boot. Each SECURE-created backup tape contains both directory information and text for the files on that tape, so to recatalogue specific files, either the backup tape or the MFD snapshot tape may be used. To both recatalogue a file and load its texts in a single operation, a LOAD FROM IBACKUP (with IBACKUP equated to the backup tape) operation should be done.

7.3. DISK PREPPING

Three prep factors are allowable when prepping disk packs. The standard prep factor is 112 words/record, (4 sectors/record). The 56 words/record (2 sectors/record) and 28 words/record (1 sector/record) prep factors are allowable, but should only be used on special applications. (Application studies will determine this.)

The data capacity of a disk track is decreased as the size of the records is decreased. However, if a significant proportion of the I/O writes to a disk are less than 56 words (or 28 words), the smaller record sizes should be used. This will increase efficiency by decreasing the size of the I/O transfers and by decreasing the size of the EXPOOL buffers necessary for storage of the partial beginnings and partial ends.

7.3.1. DPREP 1100

7.3.1.1. General

DPREP 1100 is an integral part of the Peripheral Test Sequence (PTS) diagnostic software in both online and offline modes of operation. A disk pack may be prepped by DPREP 1100 and then utilized by the Executive. If a disk pack is suspected of being defective, the diagnostic routines are able to utilize the disk pack in a way that will not affect customer files.

The benefit to a new system is the elimination of prepping during cold starts. Once the installation is complete, the diagnostic tests have been run, and the system is validated, the Executive is able to initial boot on pre-prepped packs.

The benefit to an established system is the ability to prep packs at a batch or demand level rather than Executive worker level. This has less impact on system performance and channel utilization. It means that diagnostic packages are able to identify errors with greater ease by using both the disk drive and the disk pack that encountered the error. It means that once diagnostic tests have been run, a disk pack may be used by the Executive without a costly re-prep.

7.3.1.2. Online DPREP 1100

The DPREP 1100 package contains eight different features. The software does an absolute assign of a reserved (RV) disk unit. Hence, in order to initiate a prep the pack must be mounted on a serviceable unit that is in the reserved (RV) state. The absolute pack assignment configuration parameter P3PREP must be turned on during the EXEC system generation. Also SYS$*RUN$ must contain the runstream to start the prep and SYS$*LIB$ must contain the DPREP 1100 absolute element. There are three unique DPREP 1100 absolute elements for each of the three major groups of 1100 Series systems. DPREP80 is for the 1100/80, DPREP10 for the 1110, and 1100/40, and DPREP8 for the 1108, 1106, 1100/10, and 1100/20. The runstream can then be started to perform the prep.

Eight types of prep features are available in the software. These features are:

- Type 1 WRITE HOME ADDRESS

  The purpose of this feature is to write home addresses on a pack which has no formatted tracks (pack contains only index marks or the home addresses have been obliterated). This prep writes only the home address on each physical disk track (no records are formatted for EXEC use). Performing this prep on an EXEC formatted pack will destroy the data information contained on the pack but will not destroy any bad track flags. Warnings are provided to prevent inadvertent destruction of the data.

- Type 2 PRINT BAD TRACK HISTORY

  Prerequisite:

  > The pack must be a factory certified pack or previously had prep Type 3 or Type 5 performed. (Type 3 is unavailable with the 5046 control unit.)

  A hard copy listing is provided designating the tracks flagged defective and the origin of the bad track flag. There are three types of flags for defective tracks; user flag (EXEC), factory flag (FCTY), and PTS flag (PTS) for diagnostic tests. This feature performs read only functions and does not alter any information on the pack.

- Type 3 DROPOUT OR SURFACE ANALYSIS

  This routine will destroy the data contained on the pack. Warnings are provided to prevent inadvertent destruction of the data. This feature writes all ones for a complete physical track surface, reads the data and verifies that surface will support customer data. The tracks which fail during this surface analysis are automatically flagged defective. A printout is provided of the tracks flagged defective. Surface analysis is not performed on tracks previously flagged defective.

Type 3 prep is not available with the 5046 control unit.

- Type 4 NEW PREP

  Prerequisite:

  Defective tracks should be previously flagged by either factory certification or Type 3 prep. If this type is run to completely re-prep a pack which had been previously prepped, the bad tracks information is valid and not altered.

  Warnings are provided to prevent inadvertent prep of a previously EXEC formatted pack. This prep type requests information, via the console, concerning the pack label, prep factor, location of the directory track, and additional directory tracks. The pack is completely re-prepped in the requested format without destroying the bad track history.

  Note that when a track is to be prepped, a read of the home address and track descriptor record (R0) is done. Only if the above was read without encountering the defective track indicator in the home address would the prep of the track actually take place. A re-prep of a defective track that had been flagged by the manufacturer will never be allowed.

  Also, when a track is flagged as defective, the software writes the R0 with an 8-byte data field specifying "EXEC" in ASCII. This allows a re-prep of a software marked defective track to be possible and would uniquely identify those tracks marked as defective by the manufacturer and by the software. In addition to writing the home address and R0, the remaining portion of the track is erased. Also, the EXEC hardware and software bit maps are updated to reflect the bad track.

- Type 5 PARTIAL PREP

  Prerequisite:

  The pack must be previously prepped in EXEC format.

  This prep is used to up or down a physical disk track. Parameters are solicited via the console requesting the decimal cylinder and head (physical track) to be altered. The track is then re-prepped in the same format as the other tracks on the pack or marked defective. The EXEC directory track information is updated and re-written on the pack at the original location. The prep allows one physical track to be designated per console request and the prep terminates when an 'N' keyin is received.

  Disk track repreps are inhibited under the following conditions:

  a. A keyin requests a track to be upped that was flagged defective by factory certification.

  b. Customer data resides in the affected area.

c. An unsuccessful reprep of the track is signified by the control unit.

d. Physical track 0 (VOL1), the directory area, or a reserved EXEC area is to be altered by the reprep.

If the reprep is not allowed, a console message will indicate the reason.

■ Type 6 PREP VERIFY

Prerequisite:

The pack must be previously prepped in EXEC format.

This routine performs data write, read and verify by FASTRAND track format in areas where no customer data resides. For areas where VOL1, Directory Track or customer data resides the routine performs two reads, comparing the first read with the second read for data verification. This routine may be used on any previously prepped EXEC format pack. The prep verify does not alter the pack data nor does it mark tracks defective if the data has a miscompare. Type 5 prep may be used to down any tracks found defective by this routine.

■ Type 7 DRS PREP

Prerequisite:

This routine must follow Type 4 prep to format the pack for the Disk Resident System (DRS).

This routine allows a DRS prepped pack to be generated. The pack is constructed with a 02000 word Initial Program Load (IPL) block placed on cylinder zero, head zero. The format of the track is:

I	HMA	G	R0	G	R1	R1 Data (02000 Data) IPL Block	G	R2	G	VOL1 R3 Data

where:

I	Index Mark
HMA	Home Address
G	Gap
R0	Track Descriptor
R1	Record One
IPL	Initial Program Load
R2	4-Byte Record
R3	VOL1 Label Block

- Type 8 PRINT VOL1 AND DIRECTORY TRACK

  Prerequisite:

  The pack must be previously prepped in EXEC format.

  The feature provides a printed listing of VOL1 and the first directory track. The data on the pack is not altered.

7.3.2. Recommended Usage

This subsection describes the recommended usage of DPREP 1100 for the more common situations.

7.3.2.1. Prepping a New Pack

In order to prep a pack that has not been previously prepped by DPREP 1100 (new pack):

1. Perform Types 1, 3, and 4 prep. (Type 3 is not available with 5046 control unit.)
2. If a list of known defective tracks is available also use Type 5.
3. Perform Type 2 to obtain a new list of defective tracks

7.3.2.2. Prepping Previously Prepped Packs

In order to prep a pack previously prepped by DPREP 1100:

1. Perform Type 4 prep
2. If a list of known defective tracks is available also use Type 5
3. Perform Type 2 to obtain a new list of defective tracks

7.3.2.3. Prepping a New DRS Pack

In order to prep a DRS pack that has not been previously prepped by DPREP 1100 (new pack):

1. Perform Types 1, 3, and 4 prep. (Type 3 prep not available with 5046 control unit.)
2. If a list of known defective tracks is available also use Type 5. Note that if defective tracks exist in an area to be reserved for DRS usage, a Type 7 prep will be inhibited
3. Perform Type 7 prep
4. Perform Type 2 prep to obtain a new list of defective tracks

7.3.2.4. Previously Prepped DRS Pack

In order to prep a DRS pack that has been previously prepped by DPREP 1100:

1. Perform Type 4 prep
2. If a list of known defective tracks is available also use Type 5. Note that if defective tracks exist in an area to be reserved for DRS usage, a Type 7 prep will be inhibited
3. Perform Type 7 prep
4. Perform Type 2 prep to obtain a new list of defective tracks

7.3.2.5. Defective Track Reprep

Reprep of previously suspected defective track:

1. Perform Type 5, upping suspected track

7.3.2.6. Downing of Defective Track

1. Perform Type 5, downing defective track
2. Perform Type 2 to obtain a new list of defective tracks

7.3.2.7. Testing for Additional Defective Tracks

In order to test a prepped pack that is suspected of having additional but available defective tracks:

1. Perform Type 6 prep. Note that this will not destroy the pack contents.

7.3.2.8. Reading Suspected Area of Pack

In order to read a suspected area of a prepped pack and/or to read VOL1 and first directory track of a prepped pack:

1. Perform Type 8 prep and, if desired, supply the cylinder and head numbers of the area to be read.

7.3.2.9. Surface Analysis

In order to do a complete surface analysis of pack which is suspected of being unable to retain data:

1. Perform Types 1 and 3 prep
2. If pack is usable restore with Type 4 prep

NOTE:

Surface analysis is not available with the 5046 control unit.

7.3.3. Runstream

A runstream is contained in SYS$*RUN$ and may be started by a ST keyin as follows:

```
ST   DPREP
```

In order to initiate a prep, the runstream contains a call to @DPREP. Since @DPREP (in SYS$*LIB$) does an absolute *devnam* assign, the pack must be mounted on a reserved (RV) unit and the account number/userid must be correct to pass the absolute assignment checks of the Executive.

Prior to allowing any absolute assignment of a previously prepped pack the operator must respond with a 'Y' to the following console message:

```
0 - ADH ACCESS OK RUN (runid) PACK (pack-id) YN
```

Hence operator approval and the correct account number/userid is required to use any of the features.

7.3.4. DPREP 1100 Messages

For a detailed summary of DPREP messages see SPERRY UNIVAC 1100 Series Executive System, Operator Reference, UP-7928 (current version).

7.3.5. Wall-Clock Time for Prep Operation

The following wall-clock time approximations for prep operation is dependent upon DPREP 1100 being the only activity in the system. The timings will vary depending upon the number of active runs in the system as DPREP is working in user mode. The timings will also vary depending upon the type of disk used. These figures are approximations for 8414, 8424, 8425, 8430 and 8440 disk packs. The 8433 preps take a longer time and the 8405 preps are considerably shorter.

5024/5033 CU		5046 CU	
Type	Minutes	Type	Minutes
1	15	1	10
2	10	2	7
3	20	3	NA
4	20	4	6
5	1 per keyin	5	1 per keyin
6	40	6	20
7	1	7	1
8	2 per keyin	8	2 per keyin

7.3.6. DPREP Detailed Information

Detailed information concerning the internal construction of the DPREP package may be found in SPERRY UNIVAC 1100 Series Peripheral Test Sequencer, Diagnostic Test Description DA3004. Detailed information concerning the offline usage of the Prep package may be found in DA3009. DA3004 and DA3009 are Customer Engineering Publications.

7.4. SYSTEM FILES

7.4.1. SYS$*RUN$

The file SYS$*RUN$ is created by the canned run, SYS, during a boot with Jump Select switch 4, or Jump Select switches 4 and 13 set. SYS$*RUN$ is the 5th file on the boot tape which is in COPOUT format. RUN$ is a program file which consists of a number of symbolic and absolute elements.

The one element which must be present in this file is the symbolic element called BOOTELT. This element is a partial runstream which is added (@ADD) and executed as part of the SYS run. Its functions in the SYS run are:

1. to catalogue and load SYS$*RLIB$ from the 6th file on the boot tape if a JK 4 or a JK 4 and 13 boot is taking place,

2. delete the old SYS$*SYS$MAP and catalogue and load from the 2nd or 7th file on the boot tape a new SYS$*SYS$MAP if a tape boot is taking place, and

3. call SECURE to list all disabled files if a recovery boot is taking place.

The determination of the type of boot taking place is made by testing the condition word which is set-up by a @SETC control statement in the beginning of the SYS run. The SETC control statement is one of the 'canned' control statements existing for the SYS run which exists in the element INDRIV. Before starting the SYS run, INDRIV modifies the canned SETC control statement by setting up the 'value' part of the control statement to reflect the type of boot operation taking place. The following is a list of the condition word values and their associated meanings:

1. S3
   - 00 – Drum disk boot
   - 01 – Tape boot

2. S4
   - 02 – Jump key switch 13 and 4 boot
   - 01 – Jump key switch 4 boot
   - 00 – Neither Jump key switch 4 or 13 set.

Other elements in SYS$*RUN$ are:

1. LOGFED

   the absolute form of LOGFED.

2. LOGFED

   a runstream which contains a call to LOGFED.

3. FIREUP

   a runstream which catalogues and loads the 1100 Test Package tests.

4. DPREP

   a runstream which initiates a disk prep.

7.4.2. General EXEC File (GENF$)

SYS$*GENF$ is a general EXEC file and it is a catalogued file thus allowing recovery of the file. The file contains the following items.

1. Symbiont output queue
2. The scheduling queue
3. The temporary log queue
4. Facility INFOR statements for batch runs which are presently undergoing facility synopsis or which are in a facility wait state.
5. Spooled output for demand runs.

8. Resource Control

8.1. 1100 SERIES OPERATING SYSTEM SECURITY

8.1.1. Introduction

The purpose of any security system is to limit access to any system component to those possessing the appropriate credentials.

Sperry Univac assumes that the ultimate responsibility for system security rests with the individual site. This is especially true in the areas of remote communications, computer operations, and the control of physical access to equipment. It is Sperry Univac's objective to provide an operating system which satisfies the general community's need for security. This system is expandable through programs, options, and operating procedures to attain the degree of security required.

The system will only work as desired if a security-conscious attitude is developed among all users of the system and the operating personnel. Each person must understand his or her role in the system and realize that carelessness or the failure to report known violations can jeopardize system security.

8.1.2. Current Features and Capabilities

Many features are provided which allow an installation or individual user the capability of providing for the privacy and validity of programs, files, and other data. The following summarizes many of the features available to the site manager.

- TSS

  The Terminal Security System (TSS) provides access validation of demand terminal users by verifying a unique user-id/password combination as part of the terminal log-on procedure. Access validation of batch runs is also provided. TSS will provide automatic @RUN card generation and by specification can limit users to specified execution modes.

- Keys

  Access to catalogued files can be restricted with read and/or write keys. The ability to easily change a file's keys is provided. This, together with the provisions to declare a file as private, can be effectively used to limit file access.

- MSCON$

  MSCON guarantees to obscure vital information in MFD directory items obtained by a nonprivileged user. Any future extensions to the MSCON$ function set will continue to require the calling run to be privileged if necessary.

- Tape Labeling

  The tape labeling package provides the user with the capability of protecting data contained on magnetic tapes. New implementation in this area will adhere to recognized standards.

- Guard Mode

  All user programs run in user mode with guard mode storage protection. As many system components as possible, including system processors, run in user mode. The EXEC controls shifting between user and EXEC mode.

- Limits

  The EXEC controls all transfers between storage and online storage devices. Storage limits are checked prior to initiation of each data transfer to and from a program area.

- Facilities

  Facility assignment and storage allocation are controlled by the EXEC.

- SECURE

  The SECURE processor provides the ability to easily produce a backup copy of any user defined file.

- Main Storage

  The ability to clear main storage prior to program loading or expansion is provided. The use of this feature is configurable (CONFIG tag CLRCOR equals 1).

- Offline Hardware

  Classified data which resides on fixed mass storage may be rendered inaccessible when the system is operating without full security protection by either taking those units offline or by using the SECURE processor to REMOVE critical files.

- Logging

  An extensive logging mechanism exists to record all significant events which occur during system operation. The system log file can be used to locate attempted security violations. For instance, current log entries provide an effective trace of mass storage file references.

- Privileged Mode

  Runs can assume a privileged status by assigning the file SYS$*DLOC$ with the correct keys and in this way gain the ability to assign any user file. The current definition of 'privileged' mode may be separated into a number of distinct privileged capabilities which can be assigned to certain users through the Quota System.

A run in privileged mode is one which has the file SYS$*DLOC$ assigned to it with the correct keys. This assignment indicates that a program within the run (such as the SECURE processor) can access the text and full directory information for all catalogued files in the system (except those with the 'G' option), without supplying any keys and without regard to whether the file is to be catalogued public, private, read-only or write-only. This capability must be used very discriminately because of the system security override involved.

- Quota System

  This feature allows recognition of every user in the system, including privileges, if any, and quota limits.

8.2. TERMINAL SECURITY SYSTEM (TSS)

8.2.1. General

The Terminal Security System (TSS) provides three general levels of demand terminal security and one level of batch mode security. Therefore, except for a few exclusions (discussed later), all runs entering the system are security checked by TSS.

TSS consists of a processor named TSS and an EXEC element named VALA. The processor is used by the site manager to build and maintain the TSS file. The validation of user-ids and passwords is performed by VALA as runs enter the system. The TSS processor may be executed from either a batch or a demand run. Batch operation will usually be desired for initial creation or large updates to the file.

The basic concept of TSS is that each user (batch and demand) is assigned a unique (to the user) USERID and a corresponding PASSWORD. The user is configured into the system by the site manager via the FORM$ TSS processor command. This USERID and PASSWORD must be presented to the operating system before the run is allowed to be scheduled for execution. In demand mode, the USERID and PASSWORD are presented via the log-on sequence where the Operating System solicits the USERID and PASSWORD from the user. In batch mode, the user includes the USERID on the @RUN card and includes a @PASSWD card somewhere in the runstream. The site manager is provided the capability via the processor to specify the action to be taken when a security violation is recognized. These choices range from letting the user execute to terminating the terminal. Also, an onsite console or terminal message can be printed in these cases.

TSS provides three general levels of security for the demand user. They are basic security, run security, and execution security. Basic security requires the user to enter a USERID and PASSWORD after which a @RUN image may be entered. Run security requires the user to enter a USERID and PASSWORD after which the Executive will automatically generate a @RUN image using the parameters specified on the FORM$ processor command. Execution security implies run security with the additional feature of placing the run in an execution mode as specified via the use of the START command.

The TSS processor is invoked by the call statement:

```
@TSS
```

followed by the TSS commands. The processor is terminated by the EXIT command or the next Executive control statement.

The administration of a large number of users at an installation becomes a very burdensome task for one site manager (holder of the master key). TSS allows the site manager to designate submasters.

The submaster acquires many of the privileges of the master when using the TSS processor commands. This allows the site manager or master to delegate authority to submasters thereby creating a hierarchy of control which may be quite natural at a large installation. Submasters are allowed to configure new users into the system (specify USERID and PASSWORD), to update those users under their control and to create new submasters which are under their control. In this way, a hierarchy of control can be established (see Figure 8–1).

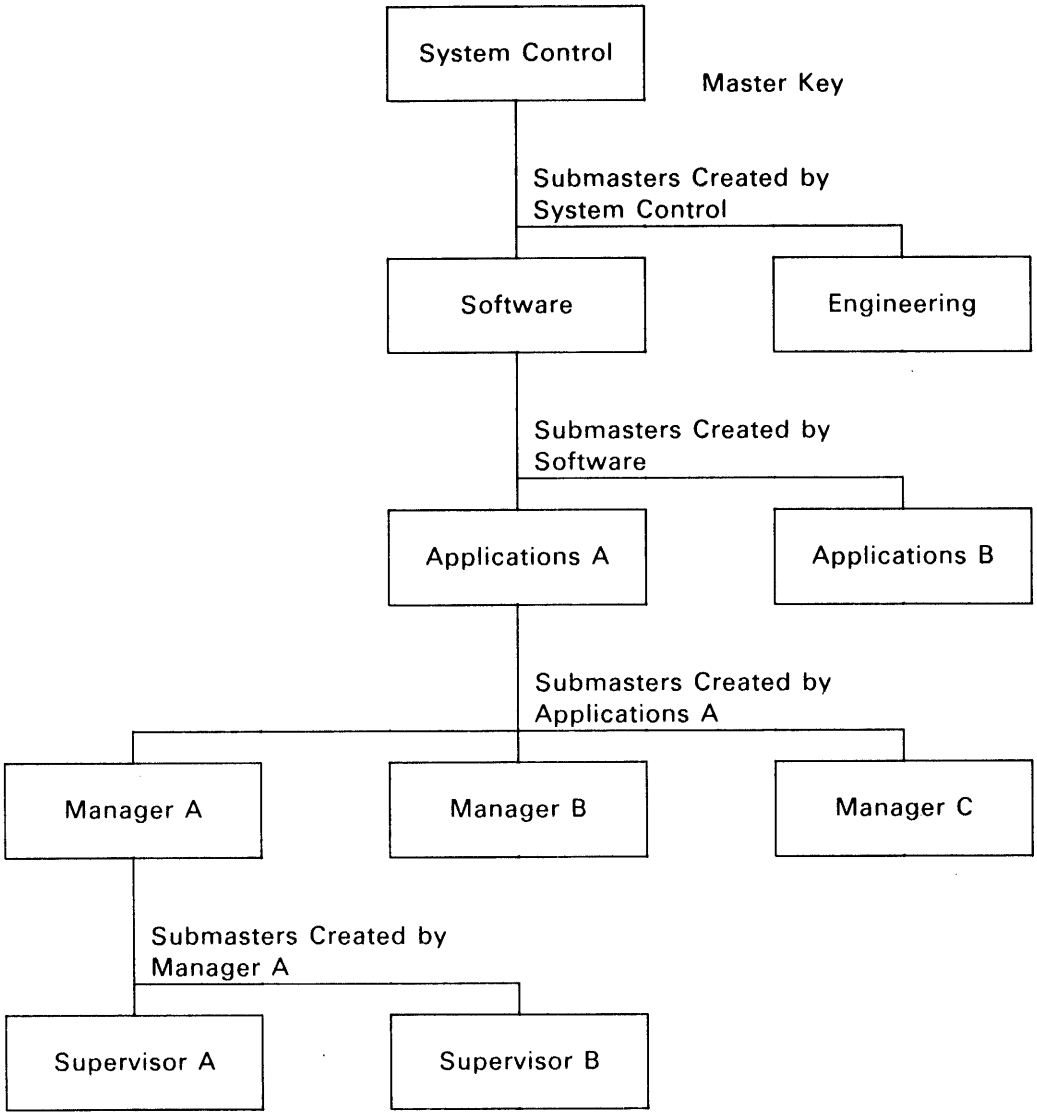


Figure 8–1. Example of Submaster Hierarchy

8.2.2. Description of Processor Commands

The TSS processor provides five types of commands. They are the master key commands, the file construction and maintenance commands, a system contingency command, the submaster commands, and a set of miscellaneous commands. All of the commands described below are privileged commands (require master or submaster key) except where noted.

The commands available via the TSS processor are:

LOCK$	FORM$	MAXTIME
OPEN$	DENY	MAXPAGE
NKEY$	REPLACE	RESET
SUBMASTER	ADDACCOUNT	LIST
NEWSUB	START	EXIT
SUBACNT	NEWPASS	MSG
CONT$	CONSMODE	LISTALL
MSGMODE	LISTSUB	

TSS commands have two major sections:

- The command field
- The operation subfields

  The period ('.') is treated as a valid character in any of the operation subfields.

  The asterisk ('*') has a special meaning only in the first, third and fourth operation subfields on the FORM$ command, otherwise it is treated as any other valid character.

  The semicolon (';') on a command indicates continuation to another image. Since it is also treated as a subfield terminator, an operation subfield may not be continued onto the next image.

  The comma (',') and space are treated as subfield terminators.

  The space period space (' . ') is treated as the command terminator.

  The '@' and '/' are treated as invalid characters. As soon as an invalid character appears in the command, it will error terminate. The following is an example:

  ```
  SUBACNT  SUB/KEY,ACT1,ACT@2
  ```

  TSS will respond with the following:

  ```
  INVALID CHARACTER '/' WAS FOUND, COMMAND IGNORED
  ```

  From the example it is clear that only the first invalid character is detected.

The following is an example of a valid TSS command:

```
FORM$  UID,PS.WD,PID,ACT1,ACT2,ACT*3
```

8.2.2.1. Master Key Commands

TSS maintains a master key in the TSS file. Whenever TSS is used (except for the limited use of the unprivileged commands available to the user), the master or submaster key must be given to TSS via the OPEN$ command. When the TSS file is initially being created, the LOCK$ command specifies the master key to be inserted in the file. In fact, even if TSS is turned on (LOGONS = 1), user-id/password validation will not be enforced until the file is created via the LOCK$ and associated commands. The master key may be changed via the NKEY$ command. If a submaster is opening the file, the key submitted via the OPEN$ command will be a submaster key which is also maintained in the TSS file.

LOCK$ Command

The format of LOCK$ is:

LOCK$ *key*

where *key* is the master key consisting of up to 12 Fieldata characters and may not contain the characters '@' 'space' or ',' (comma). Provided the file has been established via LOCK$, any additional LOCK$ commands will result in the processor being terminated with the message 'MASTER OR SUBMASTER KEY IN ERROR'.

Examples of legal master keys are:

```
MASTERKEY
123456789
ABCDEF123456
$A2C%
?
```

OPEN$ Command

The format of OPEN$ is:

OPEN$ *key*

where *key* is the previously defined master key (see LOCK$) or a legal submaster key (see SUBMASTER command). Submission of the OPEN$ command allows the user to use the privileged commands of the TSS processor.

NKEY$ Command

The format of NKEY$ is:

NKEY$ *key*

where *key* is the new master key. Only the user who has created the file with LOCK$ or opened file with OPEN$ may use NKEY$ to change the master key.

8.2.2.2. SUBMASTER Command

The SUBMASTER command is used for the creation and maintenance of the submaster hierarchy. As described in 8.2.1, submasters are allowed the use of certain privileged commands. They may use all of the TSS processor commands with the exception of LOCK$, MSG, NKEY$ and CONT$ commands.

Additionally, attempts by a submaster to use the REPLACE, DENY, ADDACCOUNT, LIST, and START commands for a user-id which does not exist in the submaster's hierarchy chain will not be allowed. A submaster can open the TSS file with a submaster key established via the SUBMASTER command. When a submaster desires to open the TSS file, the submaster key is used on the OPEN$ command.

The format of the SUBMASTER command is:

SUBMASTER *key* [,SUBMASTER]

where *key* is a submaster key of up to 12-characters which may not include the '@' 'space' and ',' characters. The word SUBMASTER following *key* is optional and, when used, indicates that this submaster key may create other submasters.

8.2.2.3. System Contingency Command

The CONT$ command specifies the contingency action to be taken by the system when an illegal user-id and/or password is used by a batch or demand user. One important feature of the CONT$ command is that a site which desires automatic run card generation but doesn't need password security can configure the system such that all passwords are legal. CONT$ can be used only by the possessor of the master key and it specifies the contingency actions to be taken for the whole system.

The format of the CONT$ command is:

CONT$ *sub-command 1,sub-command 2,....,sub-command 8*

where *sub-command* is one of the following:

ALLID	accept any USERID as legal.
CONID	when an illegal USERID is encountered, display the message 'USERID ERROR AT SITE *site-id*' at the system console.
TERID	when an illegal USERID is encountered, display the message 'ID NOT ACCEPTED' at the user's terminal.
OFFID	when an illegal USERID is encountered, terminate the user's terminal.
ALLPS	accept any PASSWORD as legal.
CONPS	when an illegal PASSWORD is encountered, display the message 'USERID *user-id* IN ERROR AT SITE *site-id*' at the system console.
TERPS	when an illegal PASSWORD is encountered, display the message 'ID NOT ACCEPTED' at the user's terminal.
OFFPS	when an illegal PASSWORD is encountered, terminate the user's terminal.

Multiple sub-commands may be used on one CONT$ command. For example:

CONT$ CONID,TERID,CONPS,TERPS

A CONT$ command with no sub-commands has the effect of clearing or unsetting any previously set sub-commands.

The default case of CONT$ (no CONT$ command executed) results in no subcommands being set. The result of this is that only configured user-ids and passwords will be accepted, no console or terminal messages will occur when unconfigured user-ids and passwords are encountered and the terminal will not be terminated.

The hierarchical relationship among the OFFID, TERID, ALLID, OFFPS, TERPS, ALLPS is as follows:

OFFID takes precedence over TERID which takes precedence over ALLID which takes precedence over OFFPS which takes precedence over TERPS which takes precedence over ALLPS.

8.2.2.4. File Construction and Maintenance Commands

The TSS processor provides the FORM$, DENY, REPLACE, ADDACCOUNT, START NEWPASS, MAXTIME, MAXPAGE and RESET commands to be used in constructing and maintaining the TSS file. These commands deal with the user information maintained in the TSS file. Userids may be 12 characters in length and passwords may be six characters in length.

FORM$ Command

The format of the FORM$ command is:

FORM$ [*1] USERID,PASSWORD [,[*2] PROJECTID, [*3] ACCOUNT [,ACCOUNT 2] ,...]

where:

*1 — When the asterisk (*) and a project-id and account field are present on the FORM$, signifies that the user has the option of automatic @RUN image generation (run mode). At log-on time if the user wishes to enter the @RUN image, the user-id/password entry must be preceded with the asterisk (*). Otherwise the user will get automatic @RUN image generation.

When the asterisk (*) is present but the project-id and account fields are not present on the FORM$, then the user must always enter the @RUN image regardless of whether or not the asterisk preceded the user-id/password entry at log-on time.

When the asterisk is not used on the FORM$, the user will always obtain automatic @RUN image generation.

USERID — A user-id of up to 12-characters (cannot have a space, slash or comma

PASSWORD — A password of up to 6-characters which is associated with the user-id (cannot have a space, slash or comma).

*2 — When present, signifies the user may enter the project-id at log-on time. At log-on time, the user will be asked for the project-id by the message 'ENTER PROJECT'.

PROJECTID — The project-id as it will appear on the @RUN image automatically generated for the user. If the PROJECTID is preceded by an asterisk (*), the user may answer the message 'ENTER PROJECT' with a carriage return or TRANSMIT and the system will use the project-id specified on the FORM$. If no project-id is given and the project-id field does not contain an *, system standard project-id will be used.

*3 — When present signifies the user may enter the account at log-on time. At log-on time, the user will be asked for the account by the message 'ENTER ACCOUNT'.

ACCOUNT — The account as it will appear on the @RUN image automatically generated for the user. If the account is preceded by an asterisk (*), the user may answer the message 'ENTER ACCOUNT' with a carriage return or TRANSMIT and the system will use the account specified on the FORM$. If no account is given and the account field does not contain an asterisk (*), a system standard account will be used. The FORM$ may specify a maximum of five account numbers for the user. If more than one account is specified on the FORM$, at log-on time, the system will ask the user to choose an account by the message 'CHOOSE ACCOUNT INDEX'. The user answers this message with 1,2,3,4, or 5 depending on how many accounts were specified and which account is to be used. The user can be instructed as to the purpose of multiple accounts, however, the actual account number need not be known.

The FORM$ command is used by the site manager or a delegate to initially configure the users into the system and to later add additional users. The different formats of the FORM$ specify whether the user will be placed in basic mode or run mode as illustrated in the examples below. The START command is used to place a user in an initial execution mode.

The following examples illustrate the various structures of the FORM$ command:

1. Assume that the user is given a user-id of USER1 and a password of PASS1. The format of the FORM$ is:

   ```
   FORM$  *USER1,PASS1
   ```

   Since the asterisk was used, USER1 will be configured for basic security only, and will always input the @RUN card.

2. Assume that the user-id is USER1 and the password is PASS1. The format of the FORM$ is:

   ```
   FORM$  USER1,PASS1
   ```

   Since the asterisk was not used, USER1 will be configured for run security which means a @RUN card will always be generated with a system standard project-id and account.

3. Assume that the user will be configured for run security, i.e., a @RUN will always be generated for the user. The user does not have the option of generating the run card. The user-id is USER2, the password is PASS2, the account is ACCT2 and the project-id is PROJ2. The format of the FORM$ is:

   ```
   FORM$  USER2,PASS2,PROJ2,ACCT2
   ```

   Note that the asterisk preceding the user-id was not used, therefore the user does not have the option of inputting the @RUN card.

4. Using the same parameters as in 3. above but configuring the user to have the option of submitting the @RUN card, the format of the FORM$ is:

   ```
   FORM$  *USER2,PASS2,PROJ2,ACCT2
   ```

5. Assume that the user will be configured for run security but that the user will input the project-id and the parameters are USER3, PASS3 and ACCT3. The format of the FORM$ is:

   ```
   FORM$  USER3,PASS3,*,ACCT3
   ```

In this example, the user will be asked for the project-id with the message 'ENTER PROJECT'. In the same example, if the ACCOUNT field contained an asterisk (*) instead of ACCT3, the user will be asked for the account with the message 'ENTER ACCOUNT'.

6. Assume the user is configured for run security with three account numbers. The parameters on the FORM$ are USER4, PASS4, PROJ4, ACC41, ACC42, ACC43. The format of the FORM$ is:

   FORM$ USER4,PASS4,PROJ4,ACC41,ACC42,ACC43

   At log-on time the user will be asked to choose the account by the message 'CHOOSE ACCOUNT INDEX'. The reply may be 1,2 or 3. If the reply is any other character, the user will again be asked to choose the account index by the message 'INDEX OUT OF RANGE' and 'CHOOSE ACCOUNT INDEX' reappearing.

DENY Command

The format of the DENY command is:

DENY id-name

Where id-name may be a user-id or a submaster key. The DENY command will delete all information associated with id-name from the TSS file. If the file was opened with the master key, the run will be allowed to DENY any user-id or submaster. If the file was opened with a submaster key and the id-name is a submaster key, then the user-ids and/or submaster keys residing in this submaster's hierarchy will not be deleted but will reside in the hierarchy of the submaster which created the key on which DENY command was performed.

REPLACE Command

The format of the REPLACE command is:

REPLACE sub-command,user-id,old value,new value

where the sub-command is:

ACCOUNT — specifies the replacement of the 'old value' account with the 'new value' account for the specified 'user-id'.

PROJECT — specifies the replacement of the 'old value' project-id with the 'new value' project-id for the specified 'user-id'.

This command is used to change a user's account or project-id.

Only those user-ids which reside in the submaster hierarchy will be altered by this command.

ADDACCOUNT Command

The format of the ADDACCOUNT command is:

ADDACCOUNT user-id,new-acct1,new-acct2,...,new-acct5

where the new-acct1,new-acct2,..., are added to the specified user-id. A maximum of five accounts is allowed per user-id, therefore the ADDACCOUNT command will only add accounts up to the maximum of five.

START Command

The START command configures the user for execution mode. Run security must have been previously configured for the specified user-id via FORM$. Depending on the format used, START will configure the user for @CTS execution mode or any other execution mode the installation desires.

The formats of the START command are:

START *user-id*
or
START *user-id*,$

The first format configures the specified user for @CTS execution mode. The second format configures the specified user-id to be placed into the execution mode specified on the image following the START command. This image must begin in column two, since TSS will place a '@' in column one of the image.

The following example illustrates the use of the second format:

```
FORM$   USER5,PASS5,PROJ5,ACC5
START USER5,$
ΔED,U   QUAL*FILE.ELEMENT    ( Δ indicates a significant space )
```

NEWPASS Command

The format of the NEWPASS command is:

NEWPASS *user-id,password*

This command replaces the specified user-id's password with the specified password. This command may be used only by the possessor of the master or submaster which created the user-id.

MAXTIME Command

The format of the MAXTIME command is:

MAXTIME *user-id,value*

This command will replace the standard value used by TSS for max-time on the generated @RUN image with the new max-time *value* specified for the user-id. This is a non-privileged command available for use by both privileged and non-privileged users. The maximum value allowed is 999.

MAXPAGE Command

The format of the MAXPAGE command is:

MAXPAGE *user-id,value*

This command will replace the standard value used by TSS for max-pages on the generated @RUN image with the new max-page *value* specified. This is a non-privileged command available for use by both privileged and non-privileged users. The maximum value allowed is 999999.

RESET Command

The format of the RESET command is:

RESET *user-id*

This command resets the run number value for the specified user-id to zero. The run number is maintained in the TSS file and printed out at the user's terminal each time a user configured for run or execution mode logs-on the system. This is a non-privileged command available for use by both privileged and non-privileged users.

8.2.2.5. Miscellaneous Commands

Six miscellaneous commands are provided by the TSS processor and are described below.

LISTALL Command

The LISTALL command can be used by the site manager or a delegate to list the information pertinent to the user-ids in the TSS file. If the requestor is a submaster, only the information pertinent to the user-ids residing in its hierarchy will be listed.

The formats of the commands are as follow:

LIST *userid*

LISTALL *key*

where:

key is either a valid master key/submaster key or is missing. In the case where *key* is missing, the TSS file must already have been opened with a valid master key/submaster key which will be assumed to be *key*.

The first format lists the TSS file information pertinent to the specified userid. The information listed is the userid, project-id, account(s), type of execution mode, if any, max-time, max-pages, run number, time and date of last run, and console mode. Note that the password is not listed.

The second format lists the same information described above for every userid in the TSS file. A submaster may list only the information (not including passwords) associated with userids in the submaster's hierarchy chain.

LISTSUB Command

The LISTSUB command can be used by a submaster key holder to list the information which the submaster has control over such as accounts, userids (2-word entries), console mode and site-ids and message categories. Such information could be listed completely or partially, depending upon the keywords specified on the command. At least one of the four keywords (ACCOUNTS, USERIDS, CONSMODES, MSGMODES) must be specified on the command, otherwise it will be ignored and an error message will come out.

The format of the command is as follows.

LISTSUB *subkey, keyword 1 [,keyword 2] [,keyword 3] [,keyword 4]*

where:

subkey is a valid submaster key,

keyword 1 is any of the four keywords mentioned above.

keyword 2 if present is one of the four keywords different from keyword 1

keyword 3 if present is one of the four keywords different from keyword 1 and keyword 2.

keyword 4 if present is one of the four keywords different from keyword 1, keyword 2 and keyword 3.

Examples:

LISTSUB *subkey1*,ACCOUNTS

LISTSUB *subkey2*,CONSMODES,MSGMODES

LISTSUB *subkey3*,USERIDS,CONSMODES,ACCOUNTS,MSGMODES

EXIT Command

The format of the EXIT command is:

EXIT

This command terminates the TSS processor.

MSG Command

The MSG command allows the site manager to store a system message in the TSS file. Each time a demand user logs on the system, this message will be displayed at the user's terminal.

The format of the MSG command is:

MSG

text of the message

where the text starts on the image following the MSG command. The message format will be displayed exactly as it is specified by the MSG command. If the message consists of 3 lines of 20 characters each, then it will be displayed at the terminal in that format. The message must be terminated with the backslash (\). Each time the MSG command is invoked, any existing message text in the TSS file will be erased and replaced with the new message.

The message text is stored in the TSS file in a 224 word block in SDF format with ASCII characters. One SDF control word is needed at the beginning and end of the text, and one control word is needed for each line of the message. Therefore, the formula for determining if the message of the day is too long is:

$$2 + \sum_{n=1}^{L} [(c_n // 4) + 1] \leqslant 224$$

where L is the number of lines and c_n is the number of characters per line. Message text input is truncated without an error message when the block is full and further message text images will be treated as additional TSS commands. The approximate maximum size of the message text is 800 characters.

The following examples illustrate the use of the MSG command.

```
MSG
THIS IS A TEST MESSAGE
TO BE DISPLAYED
AT THE USER'S TERMINAL.\
```

The above message will be displayed at the terminal exactly at it appears.

The following example illustrates how the message can be deleted from the TSS file.

```
MSG
\
```

CONSMODE Command

The CONSMODE command can be used by the site manager or a delegate to specify the console mask and site-id(s) associated with a user-id. The format of the command is as follows:

CONSMODE *user-id mode site-id*$_1$*,site-id*$_2$*,....site-id*$_n$

where:

user-id is a valid user-id previously defined in the TSS file.

mode is the console mode which this user-id is allowed to use when using the @@CONS. The console mode choices are:

BASIC	Basic Keyin
LIMITED	Limited Keyin
FULL	Full Keyin
DISPLAY	Display and Keyin

site-id is one or more site-ids of up to 6 characters each which, if used, specify the site-ids that can be used as remote consoles by this user-id.

The following is an example of a CONSMODE command:

```
CONSMODE XYZ LIMITED RS01,RS02,RS03
```

Each time the CONSMODE command is executed the console site-ids specified on it are added to the list of old console site-ids in the TSS file for the user-id. The old console mode is also replaced by the one specified on the command. If the requestor is a submaster, only those site-ids which the submaster has control over will be added to the old list. The same thing holds true for the console mode. The CONSMODE data in the TSS file can be deleted by using the CONSMODE command with only the user-id field and no mode or site-id field. This form is as follows:

CONSMODE *userid*

The site-id field is optional and is used only when a user-id is to be allowed console mode on one or more specific site-ids. If the site-id field is not used, then the user-id is allowed to use console mode on any site-id.

Any number of site-ids may be specified for any user-id on a CONSMODE command. The semicolon (;) can be used on the command if the input is to be continued on the next image.

The user-id used on the CONSMODE command must have been previously defined in the TSS file via a FORM$ command. Use of the CONSMODE command on an undefined user-id will result in the command being rejected.

When a user-id is restricted to specific site-ids, file space in the TSS file is allocated for one to n site-id buffers which are chained to the user-id record. Each site-id buffer can hold up to 26 site-id names. This file space will be used to hold the site-ids.

MSGMODE Command

The MSGMODE command can be used by the site manager or a delegate to specify the console message categories which will be received by this user-id when in the Display and Keyin mode. A maximum of 8 message categories can be specified. The format of the command is as follows:

MSGMODE *user-id* $group_1$,$group_2$,....$group_n$

where:

user-id is a valid user-id previously defined in the TSS file

group is the console message category name which can be a maximum of 6 characters.

The following is an example of an MSGMODE command:

```
MSGMODE XYZ LIST1,LIST2,LIST3
```

Each time the MSGMODE command is executed, message modes specified on it are added to the list of old message modes in the TSS file for the user-id.

If the requester is a submaster, only those message modes which the submaster has control over will be added to the old list.

A maximum of 10 message modes are allowed to be in the TSS file for a user-id. If an attempt is made to exceed this, the excess modes will be ignored.

The MSGMODE data for the user-id can be deleted from the TSS file by using the MSGMODE command with only the user-id field and no message groups. This form is as follows:

MSGMODE *user-id*

The semicolon (;) will be used on the MSGMODE command if the input is to be continued on the next image.

The user-id used on the MSGMODE command must have been previously defined in the TSS file via a FORM$ command. Use of the MSGMODE command on an undefined user-id will result in the command being rejected.

8.2.3. Use of the Execution Mode

The execution mode, initiated via the START command, can be used as a convenience to the user or as a security measure. As a convenience measure, the user will always be placed in a specified execution mode when logging on the system. As a security measure, the site manager should be aware that a user can be 'forced' into a specified execution mode, however the user can terminate that mode and then be free to do anything as long as the run is not @FIN'd. Of course, if the run is @FIN'd, the user will again be 'forced' into the specified execution mode.

8.2.4. TSS File Structure

The TSS file is divided into three parts: the header, the fixed portion, and the allocatable portion. Each part is discussed in the following subsections.

8.2.4.1. TSS File Header

The file header consists of eight sectors. Sectors one through seven are unused. Sector 0 has the format shown in Table 8–1 and described below.

Table 8–1. TSS File Header

0 1	master key	
2	system options	
3	reserved	unused
4	reserved	
5	spaces	
6	spaces	
7	NOBS	sector address of message of the day block or zero
8	reserved	
9	reserved	
10	sector address of released sector chain or zero	
11	sector address of unallocated file space	
12	sector address of released block chain or zero	
13	reserved	
14	reserved	
15	spaces	
16	spaces	
17 // 27	reserved	

Word 1

master key — The key specified on the LOCK$ command which must be provided on the OPEN$ command to execute privileged TSS commands.

Word 2

system options — The option bits are:

0 All passwords are legal.
1 Console message on password errors.
2 Terminal message on password errors.
3 Terminate on password error.
4 All user-ids are legal.
5 Console message on user-id errors.
6 Terminal message on user-id errors.
7 Terminate on user-id errors.

Word 7

NOBS — Number of sectors used for message of the day.

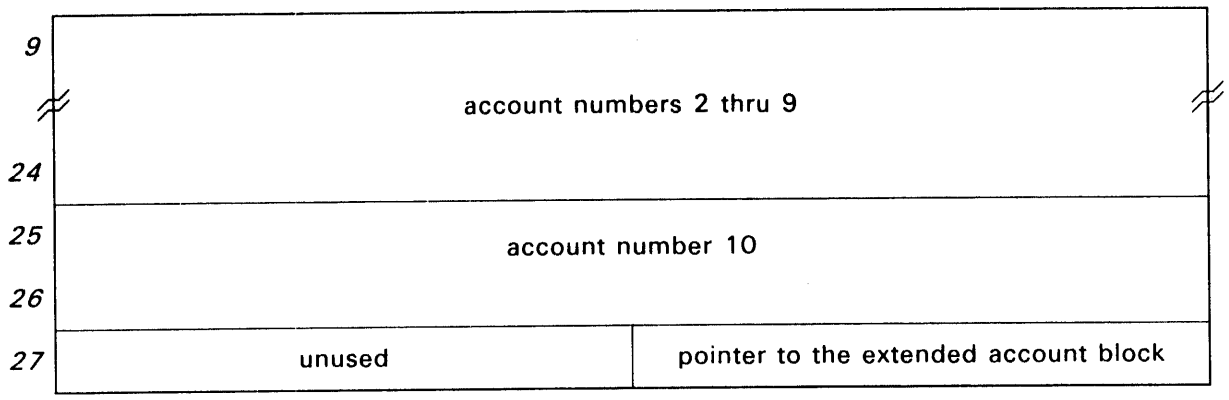
Words 10,11,12 — TSS file space is allocated in sectors or blocks (8 sectors). Words 10 and 12 are sector addresses of the beginning of a released sector chain and a released block chain respectively. Word 11 is the sector address of the beginning of the TSS unallocated file space.

The format of the submaster descriptor area is shown in Table 8-2.

Table 8-2. TSS Submaster Descriptor Area

0	submaster key			
1				
2	number of accounts			link to message group sector
3	unused	flag	console mode	link to console site-ids sector
4	level indicator			unused
5	unused			
6	user-id options			
7	account number 1			
8				

Table 8-2. TSS Submaster Descriptor Area (continued)



where:

link to message group sectors — Pointer to message group sector. Zero indicates no message groups are configured for the submasters.

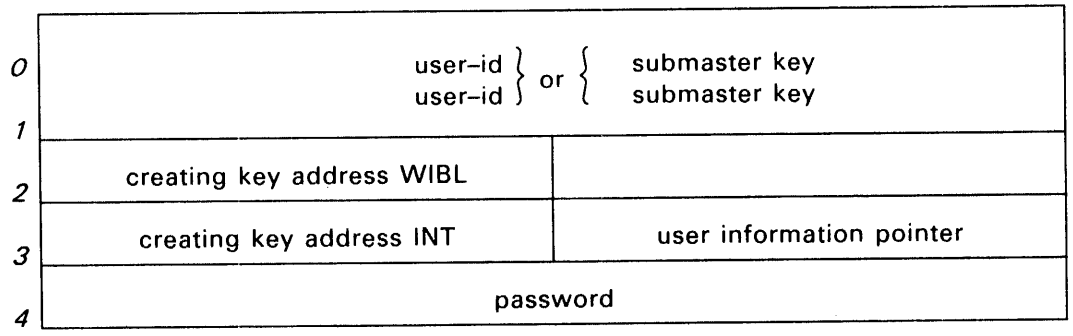
link to console site-id sector — Pointer to the first site-id sector. The site-id sectors and links for submasters are the same as for user-ids. (These pointers are sector addresses relative to the start of the TSS file.)

console mode — Zero indicates the privilege is not granted to the submasters. A nonzero console mode with a zero for the site-id sector address indicates the submaster has control over any configured site-id.

8.2.4.2. Fixed Portion

The fixed portion of the TSS file consists of 224 word blocks. The number of blocks equals REMUSE/35. REMUSE is a configuration parameter which describes the expected number of users to be described in the TSS file. Each block contains 44 five-word records, each having the format as shown in Table 8-3 and described below.

Table 8-3. Fixed Portion 5-Word Record



Word 2

H1 Integer from 1 to 44. Describes the 5-word submaster record which created this user-id record.

Word 3

H1 Sector address of the block containing the submaster record which created this user-id record. This field is zero if the user-id was created by the master.

H2 Sector address of the user information area. Each user-id record has an associated user information area (see Table 8-4).

User-ids are spread through the file by a hashing algorithm where the correct block number is calculated by dividing the sum of the two user-ids (Fieldata) by (1 + REMUSE/35) and adding 1 to the remainder because the first block is used for the file header. Note that the value for REMUSE described in the configuration is used to place the user-id entries in the file and also to retrieve them at validation time. Therefore, crossbooting to a system which is configured with a REMUSE value different from the one used for creating the file will cause legal user-id/passwords to be unrecognized at log-on time.

Table 8-4. User Information Area

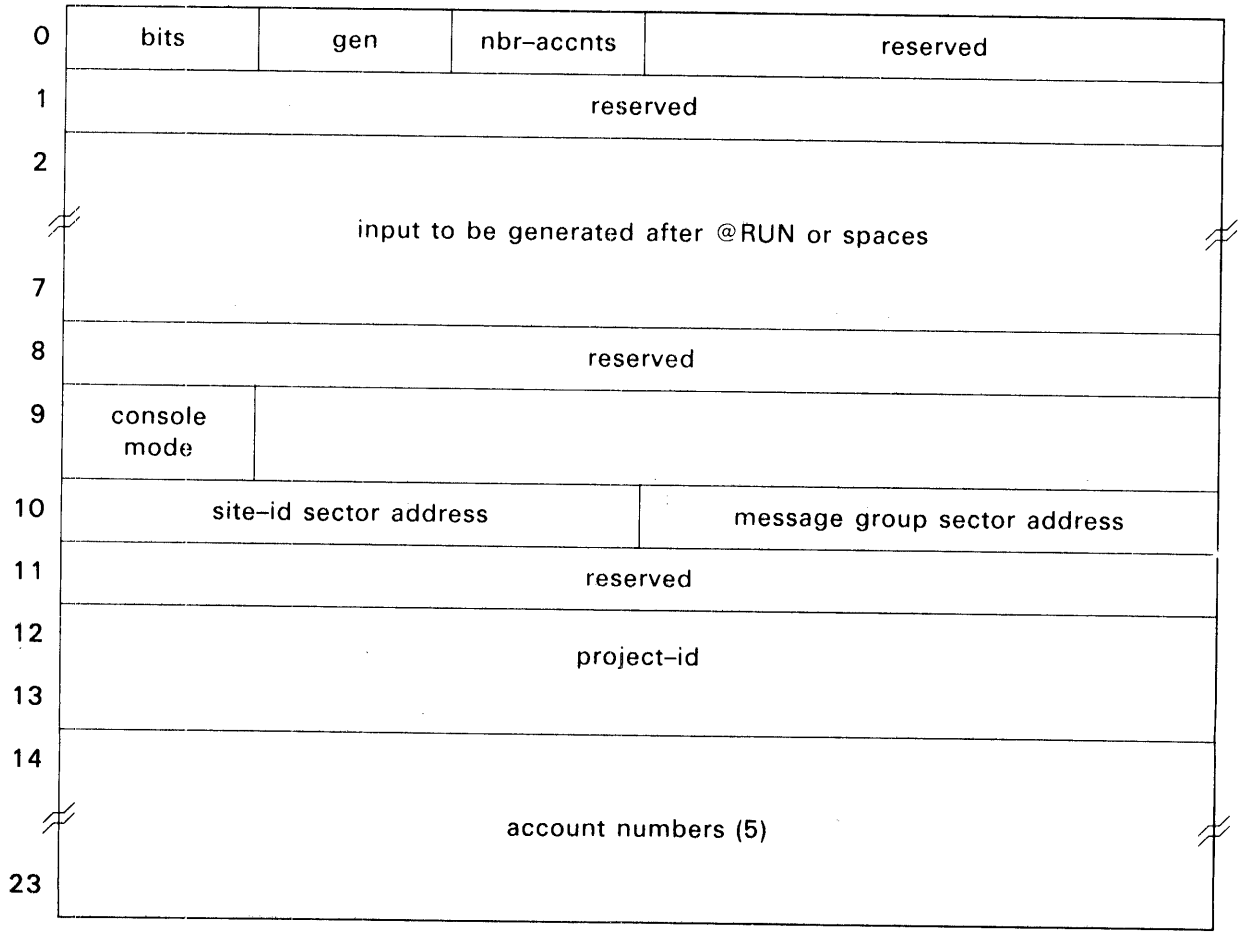


Table 8-4. User Information Area (continued)

24	max time (Fieldata)	run number
25	max pages (Fieldata)	
26	time and date of last run	
27		

Word 0

bits	35	User may supply @RUN
	34	User must enter account
	33	User must enter project-id
	32	User must enter @RUN
gen	0	No input after @RUN
	1	Generate @CTS after @RUN
	2	Generate variable input after run

Word 9

console mode	0	No Console Mode
	1	Basic Keyin Mode
	2	Limited Keyin Mode
	3	Full Keyin Mode
	4	Display and Keyin Mode

Word 10

site-id sector address	Link to site-id record buffer.
message group sector address	Link to message category buffer.

8.2.4.3. Maintenance of the TSS File

The 1100 Operating System provides the ability to save and restore certain system files including the TSS file. See 8.2.11 for a description of the save and restore function initiated from the system console.

The TSS file is assigned during the boot process before the SYS run has @FIN'd. If for some reason the TSS file cannot be assigned, the message 'TSS FILE NOT RECOVERED' will be printed at the system console. Depending on site requirements, the operator can be given instructions to perform a restore of the TSS file from tape, provided a copy of the TSS file was previously saved. If the TSS file is not restored, batch and demand password security validation will not be active and all users will be allowed to enter the system without presenting a valid password.

Since the user is allowed to change the password, the use of the save/restore feature is much more desirable in the event the TSS file is unrecoverable or if an initial boot is required. If the TSS file is recreated in these situations rather than restored from a save tape, the TSS file will not reflect the changed passwords.

8.2.5. TSS File Protection

The TSS file is assigned and catalogued during the boot process. The file is assigned exclusively to the EXEC (X option), thereby preventing any user program from assigning the file. When an initial boot is performed, it is very important to allow the run which creates the TSS file to execute before any other batch or demand runs are allowed to execute if TSS is turned on (LOGONS=1). Failure to follow this procedure could result in some user run creating the TSS file and removing control of the TSS file from the site manager.

8.2.6. Use of the Master/Submaster Hierarchy

Installations which have a large number of users should insure that the site manager is familiar with the submaster command and philosophy (see Figure 8–1). The submaster philosophy allows the site manager to delegate the authority of creating and maintaining user–id and password information. This allows each submaster to control an area of authority without gaining access to the entire TSS file. The submaster philosophy allows the master to create submasters (see SUBMASTER command) and to specify whether these submasters can in turn create submasters. This allows a complete hierarchy or chain of command to be developed where each submaster can control an area of responsibility.

8.2.7. Batch Mode Considerations

8.2.7.1. @START Runs

@START runs will assume the user–id of the run starting them and will not be required to present a @PASSWD image to the system.

8.2.7.2. @RUN,/B

A demand user entering a @RUN,/B, which schedules the run as batch, will be required to include an @PASSWD image in the runstream even though the user has already logged on the system with a valid user–id and password. The @RUN image, of course, must also have a user–id specified as the subfield of the account field.

8.2.8. TSS I/O Errors

TSS may detect an I/O error when accessing the TSS file. A detailed description of the system console messages displayed upon detection of the various I/O errors can be found in SPERRY UNIVAC 1100 Series Executive System, Operator Reference, UP–7928 (current version). The system console messages generally say 'TSS FILE DISABLED' with some additional description of which part of the TSS file was being accessed when the error occurred. If an error is detected when reading the file header portion, the Terminal Hold Condition is set, which does not allow any demand terminals to become active. Installation requirements will determine whether the operator should be instructed to remove the hold condition prior to the file being restored. The hold condition is set since, if the TSS file header cannot be read, TSS will essentially be turned off and all users will be allowed to enter the system without being validated.

8.2.9. TSS Configuration

Three configuration parameters must be set to use TSS. LOGONS must be set to one (1) to turn on TSS. If LOGONS is zero, then all users are allowed to enter the system without being validated.

REMUSE must be set to the maximum number of batch and demand users an installation expects to be described in the TSS file. This is not the number of users on the system at one time. TSS will continue to function if the number of users exceeds REMUSE, however, more efficient allocation and use of the TSS file will result if the REMUSE value is accurate. Therefore an installation need not determine an exact value for REMUSE. An approximate value sufficient. For example, if the number of users is 560, a REMUSE value of 600 is satisfactory. An installation need not reconfigure the system each time the number of users exceeds the REMUSE value.

BATPAS must be set to 1 to turn on batch mode TSS. This mode will verify user-ids and passwords for all batch runs. If set to zero, all batch runs will be scheduled without password validation.

These parameters have the following default values:

```
LOGONS  0
REMUSE  100
BATPAS  0
```

8.2.10. TSS/QUOTA Interface

If TSS is configured in the system and QUOTA has user-ids configured (USERON = 1), the installation must insure that a user-id specified for the Summary Account File is identical to the user-id specified for the TSS file for a given user. That is, when a user is assigned a user-id associated with a password to be used for TSS validation, the same user-id associated with an account number must be assigned to that user for use by the QUOTA system.

8.2.11. Summary Account and TSS File Save and Restore

8.2.11.1. General

A save/restore capability is provided by the Executive which can be used to save the TSS file and the summary account file.

The save or restore of these critical system files is initiated by a standard SV or SR console keyin. The run-id EXEC-8 uniquely identifies this special save or restore. As is standard for either keyin, the operator may also optionally indicate the specific tape equipment, device name or reel number.

For example, an operator keyin of the form

```
SV EXEC8 SYS$*ACCOUNT$R1
```

initiates a save of the summary account file. And a keyin of the form

```
SR , /reel n EXEC-8 SYS$*TSS$FILE
```

initiates a restore of the TSS master file and puts the copy on tape *reeln*.

The save or restore is accomplished using the standard software, with the following exceptions:

1. The file must be assigned to the EXEC.

2. The text of the file is checksummed, block by block, and is written on tape in 226-word blocks, including the checksum. If a checksum error occurs on a restore, the operator is notified and the restore continues.

3. The file text is also encrypted on tape according to an algorithm which is an easily-modifiable subroutine in SVKEY. The encrypted text is decoded on the restore by a matching subroutine in SRKEY. The encryption is done to render the tape useless to anyone but the authorized restore routine.

Both the summary account and TSS files should be saved periodically so that a relatively up-to-date copy of the file is always available. The need to restore the files is evidenced by the display of either of the following messages immediately after 'MASTER LOG ASSIGNED' on a recovery boot:

ACCOUNT FILE NOT RECOVERED
or
TSS FILE NOT RECOVERED

If either or both of these messages appear, the file(s) in question will have been re-catalogued and re-initialized so that a restore via the SR keyin is needed to return them to their former state. The keyin to restore either file is allowed immediately after SYS FIN.

8.2.12. Error Messages

The following is a sample of error messages presented by TSS.

ILLEGAL COMMAND

USERID SUBFIELD MISSING

SUBFIELD TOO LONG AT COLUMN 19, TREATED AS SPACES

MODE SUBFIELD NOT RECOGNIZED - The mode specified on the CONSMODE command is different from BASIC, LIMITED, FULL or DISPLAY.

LOCKED - TSS file has been initialized

USERID NOT UNIQUE

OLD SUBMASTER IS IN ERROR OR NEW SUBMASTER NOT UNIQUE

KEY NOT UNIQUE - The new key on NKEY$ command already exists in the TSS file.

OLD SUBMASTER IS IN ERROR OR FILE OPENED WITH INCORRECT KEY

USERID IN ERROR OR FILE OPENED WITH INCORRECT KEY

SUBMASTER KEY IN ERROR OR FILE OPENED WITH INCORRECT KEY

USERID OR SUBMASTER IN ERROR

WORD LENGTH EXCEEDED - Input image on a START command exceeds 7 words.

I/O ERROR COMMAND NOT PROCESSED – Read or Write failure on TSS file.

REQUEST TO READ PAST END OF SITE-ID CHAIN IGNORED

REQUEST TO READ PAST END OF MESSAGE GROUP CHAIN IGNORED

OLD ACCOUNT NUMBER OR PROJECT-ID IN ERROR

INCORRECT FORMAT FOR STATEMENT GIVEN

NUMBER OF ACCOUNTS AT MAXIMUM

SUBMASTER CREATION NOT ALLOWED

ALL ACCOUNT NUMBERS INVALID

ACCOUNT # xxxxxxxxxxx INVALID

PASSWORD SUBFIELD MISSING, COMMAND IGNORED

MAXPAGE VALUE TOO BIG, COMMAND IGNORED

MAXTIME VALUE TOO BIG, COMMAND IGNORED

ILLEGAL CHARACTER '/' WAS FOUND, COMMAND IGNORED

ACCOUNT NUMBER(S) NOT SPECIFIED, COMMAND IGNORED

TOO MANY ACCOUNT NUMBERS PRESENT, FIRST x PROCESSED

MASTER KEY IN ERROR – On the LISTALL command, the key has to be a master key.

MASTER OR SUBMASTER KEY IN ERROR – The user has attempted to use a privileged instruction.

ID NOT ACCEPTED – A user-id or password entered by the user is illegal.

USERID DIFFERENT FROM LOGON – The user has used a user-id on the @RUN image which is not the same as that used at log-on time.

MISSING USERID – The user has not specified a user-id on the account subfield on the @RUN image in batch mode.

RUN REMOVED DUE TO SECURITY ERROR – The user's run has been removed because either an illegal password or no password was found in the runstream.

8.3. QUOTA SYSTEM

8.3.1. Introduction

The Quota System is designed to accumulate statistics on and control use of system resources. The statistical accumulation can be used for billing and accounting. To provide the accumulation and control features, the summary account file maintained by the Quota System contains the following entries:

1. Basic Accounting Block

   This block consists of the account number, accumulators and limits on selected areas of system resources. In addition, a cell is provided which allows the installation manager to put a limit on the extent to which an account is allowed to utilize the system. This limit is referred to as an account-level quota and is expressed in terms of standard units of accounting (SUA). SUAs may be equated to dollars and cents, equivalent SUPs or whatever unit the site chooses. The separate accumulator values maintained during a run are converted to SUAs and accumulated into a single total used to enforce the account-level quota.

   The general formula used to calculate SUAs is:

   $$SUAs = (f_1)(r_1) + (f_2)(r_2) \ldots + (f_n)(r_n)$$

   where:

   f_n Input factor such as SUPs, track-SUPs, cards out, etc.

   r_n The rate assigned to this particular factor.

2. User-id Entries

   These entries apply only to a specific account and they identify which users may run under that account. Each user-id entry consists of the user-id, a list of privileges granted to the user, a user quota limit expressed in terms of SUAs and the user's SUAs used to date.

3. Quota Sets

   Each Quota set consists of limits defining the maximum system resources that a run can use. Also defined within the set is the type of run (onsite batch, remote batch, demand) to which the set applies and the time of day during which the set applies.

Four major components combine to form the Quota System and provide a means of manipulating the information used by the Quota System. These components are listed below along with a brief description of each one.

1. QUIP

   QUIP is a processor that provides the installation manager with an easy-to-use means to insert, modify, display and delete accounts, user-ids and Quota sets in the summary account file. This same processor can be used by any user to obtain information about the user's own account, user-id and Quota sets.

2. File Handling Code

   An EXEC element (SACCNT) and an ER (ACCNT$) provide an interface between the user and the summary account file. SACCNT controls all insertion, deletion, updating and reading of information in the summary account file.

3. Summary Account File

   The summary account file resides on mass storage and contains all accounts, user-ids and Quota sets for the Quota System.

4. Monitoring and Enforcing Code

   There are several EXEC elements which monitor the Quota limits and take appropriate action when a limit is exceeded.

The Quota System is designed to be flexible so individual sites can tailor it to their specific needs. To provide this flexibility, several areas of the Quota System are configurable. Parameters in AACONFIG allow the installation manager to configure the Quota System to the installation's needs at system generation time. The configurable areas of the Quota System are as follows:

1. The monitoring and enforcing code has been divided into five levels to provide varying levels of control.

   Level Zero

   The summary account file and all code associated with monitoring and enforcing account limits is turned off.

   Level One

   The summary account file is turned on along with the code to validate accounts. In addition, maximum priority, default priority, deadline, and real-time values are taken from the basic accounting block. All accumulation and enforcing code is turned off.

   Level Two

   Level Two is equivalent to Level One, plus the code to accumulate the usage of system resources in the basic accounting block is turned on.

   Level Three

   Level Three is equivalent to Level Two, plus the code to enforce the limits specified in the basic accounting block and the code to enforce the account-level Quota is turned on.

   Level Four

   Level Four is equivalent to Level Three, plus the code to enforce the limits specified in the Quota set is turned on.

2. For levels zero through four monitoring of the user-id as a subcategory of the account can be turned on or off. When user-ids are turned on, enforcement of the user SUA limit begins at level three.

3. For levels one through four the code to solicit the operator when an undefined account is encountered can be turned on or off. With this code turned off, users with undefined accounts and user-ids are rejected.

4. The installation manager can define groups of tapes, discs, fixed mass storage and communications/symbiont units at system generation time. Each group specifies what specific devices should be grouped together under one accumulator. For example, F17 and F4 could be grouped together under fixed mass storage devices. These same group definitions are used to specify device usage limits in the basic accounting block and in the Quota sets and are used in the log entries.

For all possible configurations, it is important to note that the enforcing and monitoring code is completely independent of the file handling code. For levels one through four, all of the file handling code is turned on.

The summary account file is exclusively assigned to the EXEC to prevent direct access of the file by users. The file handling code in the EXEC provides the only interface needed by users to reference the summary account file.

8.3.2. Quota Input Processor (QUIP)

QUIP is a processor designed to provide the installation manager with an easy-to-use interface with the summary account file. Input to QUIP consists of simple statements which direct the processor to perform certain tasks. QUIP can be used either in batch or demand mode.

Under the Quota System, runs must use a special account and user-id known as the master account and master user-id before they are allowed to perform all of the QUIP tasks. (If USERUN = 0, then just Master account.)

The runs that do run under the master account and user-id are referred to as privileged runs. Runs which are not running under the master account and user-id are only allowed to read information about the account and user-id they are running under. The master account and user-id are entered at boot time and may be changed at any time by a privileged run.

To execute QUIP, the QUIP absolute element must be in a catalogued or temporary program file. The QUIP absolute is executed by using the processor call @filename.QUIP with no parameters or by using a @XQT filename.QUIP. All input to QUIP is printed as it is read.

Allowable options are "S", which will display only source input and any error messages, and "E", which will display only source input which is in error and any error messages. If no option is specified, QUIP displays all source input, error messages, and output.

QUIP can be used by the installation manager, running as a privileged run, to insert and maintain accounts, user-ids and Quota sets in the summary account file. To perform these tasks on the summary account file, the QUIP language utilizes the commands INSERT, READ, UPDATE, ADD, DELETE, PURGE, LIST, CHANGE and END.

Before describing the syntax and usage of the commands, some special symbols used in this description and the allowable input parameters are defined. The symbols with their meanings are as follows:

[ ] the enclosed field is optional

[1] optional account parameters

[2] optional user-id parameters

[3] optional Quota set parameters

; the semicolon marks continuation of the statement

Tables 8-5 through 8-7 define the parameters QUIP can insert, update, read and delete in the basic accounting block, user-id entry and Quota sets. The tag used to identify the parameter in the table is the tag to be used as input to QUIP to reference the parameter.

When the parameters are specified as QUIP input, their format is parameter-id followed by an equal sign followed by the parameter value for the parameter-id. Spaces are allowed between the parameter-id and equal sign and are allowed between the equal sign and the parameter value. Multiple parameters are separated by spaces.

The general format of a QUIP input statement is:

command parameters

where *command* is one of the nine QUIP (see 8.3.2.1 through 8.3.2.9) commands and the parameter field specifies the account, account parameters, user-id, user-id parameters, Quota set and Quota set parameters as appropriate for the command being processed.

The format of a source image is free format; that is, the order of the parameter fields within the image is fixed, but the parameter fields are not restricted to a particular column. In addition, wherever parameters in Tables 8-5, 8-6, or 8-7 can be specified, the parameters in the table can be in any order when specified as input. Source images may continue across image boundaries by using the standard semicolon continuation character. The semicolon can be used wherever a space is allowed. When a continuation character is encountered, leading spaces on the following image are ignored. Comments may be entered following the space-period-space sequence.

Table 8-5. Basic Accounting Block Parameters

Parameter-id	Description
MP	Maximum priority allowed for the account (A thru Z). Example: MP = A
DP	Default priority for the account (A thru Z). Example: DP = K
DL	Deadline time allowed if set to one, not allowed if set to zero. Example: DL = 1
RTL	Specifies the real-time level (2-35) allowed for the account. Real time is not allowed if RTL is set to zero. Example: RTL = 25.
CEQ	If set to one, the account is allowed to exceed its maximum SUAs. Zero indicates the account cannot exceed its Quota limit. Example: CEQ = 0
SUAS	Specifies the maximum SUAs the account is allowed to use. Example: SUAS = 24000
ACMS1 thru ACMSn	Defines the maximum number of tracks of fixed mass storage the account can use for catalogued files. A maximum is specified for each fixed mass storage group that the run is allowed to utilize. ACMS1 is the maximum on fixed mass storage in group one, ACMS2 is the maximum on fixed mass storage in group two, etc. Example: ACMS2 = 400

Table 8-6. User-id Parameters

Parameter-id	Description
UQUOTA	Specifies the maximum SUAs the user-id is allowed to use. Example: UQUOTA = 2000
UCNTRL	User-id control bits. This parameter consists of 18 bits which can be specified bit by bit using the notation U1 thru U18 to specify the bits to be set. Bits 17 and 18 are used internally by the EXEC. If bits 17 and 18 are specified, they are ignored. When specifying more than one bit, the bit indicators are separated by slashes (/). The bits are numbered from left to right. Example: UCNTRL = U4/U9

Table 8-7. Quota Set Parameters

Parameter-id	Description
START	Specifies the beginning of the time interval to which the limits in the Quota set apply. The time is specified as hours and minutes using the 24-hour clock. Example: START = 1325
STOP	Specifies the end of the time interval to which the limits in the Quota set apply. The time is specified as hours and minutes using the 24 hour clock. Example: STOP = 940
RRTL	Real-time level at which a run using this Quota set is allowed to run. The level given in the Quota set overrides the level given in the basic accounting block. Allowable real-time levels are 2 thru 35. A level of zero indicates that real time is not allowed. Example: RRTL = 0
PAGES	The maximum number of pages that a run is allowed to output. Example: PAGES = 600
CARDS	The maximum number of cards that a run is allowed to output. Example: CARDS = 400

Table 8-7. Quota Set Parameters (continued)

Parameter-id	Description
SIZE	Defines the maximum program size, in blocks, that will be allowed for a run. Example: SIZE = 40
RMP	The maximum priority allowed for the run (A thru Z). The priority in the Quota set overrides the maximum priority in the basic accounting block. Example: RMP = C
CONNECT	This parameter applies to demand runs only and it specifies the maximum amount of time that a run can use a terminal. Time is specified in minutes. Example: CONNECT = 100
SUPS	Defines the maximum number of SUPs in minutes that a run is allowed to accumulate. This parameter cannot be overridden by the maximum time parameter on the RUN image. Example: SUPS = 600
VDT	Specifies the maximum voluntary delay time, in minutes, that a run is allowed to accumulate. Example: VDT = 30
D1 thru Dn	This parameter consists of two values separated by a comma. The first value defines the maximum number of removable disk units per disk group that a run can have assigned at one time. The second value defines, in minutes, the length of time each unit can be assigned. D1 defines the number of removable disk units and time per unit for removable disk units in group one, D2 defines the number of removable disk units and time per unit for removable disk units in group two, etc. Example: D1 = 1,75

Table 8-7. Quota Set Parameters (continued)

Parameter-id	Description
DES1	Descriptor word one.
DES2	Descriptor word two. Descriptor words one and two define various capabilities, limitations and attributes of the run using the Quota set. Each parameter consists of 36 bits which can be specified as one numeric value or specified bit by bit using the notation Q1 thru Q36 for descriptor one and Q37 thru Q72 for descriptor two to specify the bits to be set. If more than one bit is specified, the bit indicators are separated by slashes (/). The bits are numbered from left to right. In general, if a bit is set, the privilege is granted. The meanings of twelve bits in descriptor word one have been defined. If the bit is set it means: Q1 quotas apply to onsite batch runs Q2 quotas apply to remote batch runs Q3 quotas apply to demand runs Q4 deadline time allowed Q5 use of 'V' and 'G' options on @ASG is allowed Q6 absolute device name assign, arbitrary device assign and assignment of a communications device are allowed. Q7 Checkpoint/Restart is allowed. (Console keyin Checkpoint/Restart is always allowed.) Q8 ER RSI$ allowed Q9 SIP data reduction is allowed Q10 SIP ON/OFF allowed via ER SYSBAL$ Q11 ADH access to labeled packs is allowed Q12 All SIP functions allowed via ER SYSBAL$ Q13 HVTS Quota set Example: DES1 = Q1/Q2,Q4
RTMS1 thru RTMSn	This parameter defines, by fixed mass storage group, the maximum number of tracks of temporary mass storage that a run is allowed to use. RTMS1 defines the limit for temporary mass storage tracks on fixed mass storage in group one, RTMS2 defines the limit for temporary mass storage tracks on fixed mass storage in group two, etc. Example: RTMS1 = 150
RCMS1 thru RCMSn	This parameter defines, by fixed mass storage group, the maximum number of new tracks that a run is allowed to catalogue. RCMS1 defines the limit for catalogued mass storage tracks on fixed mass storage in group one, RCMS2 defines the limit for catalogued mass storage tracks on fixed mass storage in group two, etc. Example: RCMS2 = 100
T1 thru Tn	This parameter consists of two values separated by a comma. The first value defines the maximum number of tape units per tape group that a run can have assigned at one time. The second value defines in minutes

Table 8-7. Quota Set Parameters (continued)

Parameter-id	Description
	the length of time each unit can be assigned. T1 defines the number of tape units and time per unit for tape units in group one, T2 defines the number of tape units and time per unit for tape units in group two, etc. Example: T1 = 2,40

Scanning a given source image continues until the next command mnemonic is encountered or until the source input file is terminated. Each command is processed as it is read.

Numerical input to QUIP can be specified as either octal or decimal, with octal values indicated by a leading 0.

8.3.2.1. INSERT Command

The INSERT command is used to insert accounts, user-ids and Quota sets into the summary account file. The general format of the INSERT command to insert an account is as follows:

INSERT ACCOUNT *x* [1] [USERID *y* [2]] [SET *z* [3]]

where:

INSERT — Command

ACCOUNT — Keyword indicating to QUIP that an account or accounts are to be inserted.

x — Name of the account to insert. If multiple accounts are specified, they are separated by commas. The allowable characters for the account are as defined in SPERRY UNIVAC 1100 Series Executive, Volume 2, EXEC Programmer Reference UP-4144.2 (current version).

[1] — Any of the parameters listed in Table 8-5. If the installation manager does not supply all of the parameters in Table 8-5, the account is given predefined values known as system standards for those parameters that are not specified. At system generation the system standard account is defined in AACONFIG and it is given initial values for all of the parameters. When the system is booted and the Quota level is other than zero, the system standard account is automatically inserted into the summary account file. Then these predefined parameter values are available for all accounts for parameters that are not specified.

If multiple accounts are specified, each account receives the basic accounting block parameters specified as input.

USERID — Optional keyword indicating to QUIP that a user-id or user-ids are to be inserted under each account that is being inserted. This field can be specified whether or not user-ids are configured in the system.

y — One or more user-ids to insert. If multiple user-ids are specified, the user-ids are separated by commas. The allowable characters for a user-id are the same as those for an account.

[2] Any of the parameters in Table 8-6. System standard parameter values are used for any parameters that are not specified on the INSERT command. The system standard user-id is defined in AACONFIG at system generation time and it is given initial values for all of the parameters. The system standard user-id is inserted along with the system standard account at boot time. These predefined values are then available for all user-ids for all the parameters that are not specified.

If multiple user-ids are inserted, each user-id under each account receives the parameter values specified on the insert statement.

The Quota System is designed such that only runs running under the master account/user-id are allowed to manipulate accounts in the summary account file. However, when user-ids are configured, user-ids can be designated as submasters. A submaster is allowed to insert, update, read and delete user-ids under its account. A submaster is not allowed to create another submaster, nor can it give the user-ids it inserts any privileges that submaster does not have. A user-id is designated a submaster by setting bit U13 in the user-id control parameter (UCNTRL). By setting bit U14 in the user-id control parameter, a submaster can be given the privilege of inserting, updating, reading and deleting Quota sets for the user-ids that it inserts.

SET Keyword indicating to QUIP that Quota set parameters follow. The Quota set applies to all accounts and user-ids being inserted. A maximum of 13 Quota sets may be defined for the account. If multiple Quota sets are being inserted, the parameters for each set must be preceded by the QUIP keyword, SET, followed by the set number.

z A number between 0 and 12 identifying the set whose parameters follow. For multiple Quota set definitions, the set definitions are not required to be in numerical order. Quota sets are optional on the insert statement and they may be specified regardless of the Quota level.

[3] Any of the parameters in Table 8-7. For Quota levels one thru four, every account and user-id must have at least one Quota set. At system generation a system standard Quota set is defined in SGS statements. This Quota set is inserted as the Quota set for the system standard account and user-id at boot time. Then if an account and user-id are inserted without at least one Quota set specified, they receive the system standard Quota set as their Quota set and share it with the system standard account and user-id and any other account and/or user-id that was inserted without a Quota set. Note that the system standard Quota set is not duplicated in the summary account file. The single Quota set is shared by all those accounts. If Quota set parameters are supplied for an account and user-id, only those parameters specified as input will appear in the set. System standard values are not used for parameters that are not specified. Parameters not specified as input will appear as zero in the Quota set.

Examples:

1. INSERT ACCOUNT 123

   QUIP will insert account 123 without user-ids. The values for the basic accounting block will come from system standards and the system standard Quota sets will be used.

2. INSERT ACCOUNT 456,789 MP=B USERID ABC,DEF

   QUIP will insert accounts 456 and 789 with user-ids ABC and DEF inserted under each account. The values for the basic accounting block and user-id entry will come from system standards except for maximum priority which is set to B. The system standard Quota sets will be used for the accounts and user-ids.

3. INSERT ACCOUNT 666555 SET 0 START=0 STOP=600;
   PAGES=700 SIZE=040 RMP=P SUPS=60 DES1=;
   Q1/Q2/Q8/Q12 T1=2,40

   QUIP will insert account 666555 with no user-id. The values for the basic accounting block will come from system standards. Quota set zero is defined for this account allowing onsite and remote batch runs to run from midnight until 6 in the morning. Each run is allowed to print 700 pages, have a maximum main storage size of 040 blocks, use 60 SUP minutes and assign two tapes of the types defined in tape group one for a total of 40 minutes per assignment.

The format just described allows the installation manager to insert user-ids at the same time that an account is inserted. There is another INSERT format that allows user-ids to be inserted under an account that already exists. The format is:

INSERT USERID *y* [2] UNDER ACCOUNT *x* [SET *z* [3]]

where:

INSERT	Command
USERID	Keyword with the meaning as previously defined.
y	User-id or user-ids to be inserted.
[2]	Any of the parameters in Table 8-6. System standard values are used for parameters that are not specified. If multiple user-ids are specified, the parameters specified on the input statement are given to each user-id.
UNDER	Keyword indicating to QUIP that the account or accounts under which to insert the user-id or user-ids, already exist.
ACCOUNT	Keyword indicating to QUIP that accounts follow.
x	Existing account or accounts under which to insert all user-ids specified. Multiple accounts are separated by commas.
SET	QUIP keyword indicating to QUIP that Quota set parameters follow. This keyword must preceed the parameters for each set to be inserted.
z	A number between 0 and 12 identifying the set whose parameters follow. Every Quota set must have a set number.
[3]	Any of the parameters in Table 8-6. Quota sets are optional on the insert statement. If Quota sets are specified, they are inserted once and used by all of the user-ids being inserted. If Quota sets are not specified, the user-ids share the Quota sets being used by the account under which the user-ids are being inserted.

Examples:

1. INSERT USERID BCD UQUOTA=4000 UNDER ACCOUNT 1234

   QUIP will insert user-id BCD under the already existing account 1234. The user-id parameters will be taken from system standards except for SUA limit which is set to 4000. The user-id BCD will share the Quota sets being used by account 1234.

2. INSERT USERID EFG UNDER ACCOUNT 567,888 SET 2;
   START=2000 STOP=300 PAGES=1000 SIZE=030 RMP=;
   E SUPS=60 DES1=Q1/Q2 T2=2,30 SET 4 START=;
   1200.....

   QUIP will insert user-id EFG under the already existing accounts 567 and 888. The user-id parameters will be taken from system standards. Quota sets two and four are defined for the user-id and they will be inserted once and shared by both user-id entries. These Quota sets are only specified for the user-id EFG and they will only be used by runs under EFG.

It is possible that the Quota sets for many accounts will be exactly the same. To require each account and user-id to have duplicate copies of these Quota sets results in an unnecessary waste of file space in the summary account file and would make insertion of all these Quota sets a tedious job for the installation manager. Therefore, the concept of shared Quota sets was designed into the Quota System. This concept has already been briefly discussed in regard to accounts sharing the system standard Quota sets and it will be more fully described here.

Shared Quota sets simply mean that several accounts and/or user-ids share the same Quota sets. For the INSERT statements described to this point, sharing of Quota sets takes place by default. Sharing of Quota sets takes place in the following instances:

1. If an account is inserted without specifying a Quota set, the account, and any user-ids inserted with the account, share the system standard Quota sets.

2. If an account is inserted with user-ids and Quota sets, the account and user-ids share the Quota sets.

3. If user-ids are inserted under an existing account without specifying Quota sets, the user-ids share the Quota sets of the account.

4. If multiple accounts and user-ids are inserted with Quota sets, all of the accounts and user-ids share the Quota sets.

The installation manager can also use three additional INSERT statements to explicitly specify that accounts and/or user-ids should share Quota sets that already exist. The formats and meanings of these three statements are:

1. INSERT ACCOUNT *x* [1] [USERID *y* [2]] USING ACCOUNT *a* [USERID *b*]

   The account *x* is inserted with its own basic accounting block and user-id *y*, if specified, is inserted with its own user-id entry but they share the Quota sets used by account *a* or, if user-id *b* is specified, they share the Quota sets used by user-id *b* under account *a*.

2. INSERT USERID *y* [2] UNDER ACCOUNT *x* USING ACCOUNT *a* [USERID *b*]

   The user-id *y* is inserted with its own user-id entry under account *x*, but it shares the Quota sets used by account *a* or, if user-id *b* is specified, it shares the Quota sets used by user-id *b* under account *a*.

3. INSERT USERID *y* [2] UNDER ACCOUNT *x* USING [ACCOUNT *a*] USERID *b*

   The user-id *y* is inserted with its own user-id entry under account *x*, but it shares the Quota sets used by user-id *b* under account *x* or, if account *a* is specified, it shares the Quota sets used by user-id *b* under account *a*.

Multiple accounts and/or user-ids may be inserted with the above three formats.

When inserting an account and/or user-id using Quota sets that already exist, additional Quota sets may not be defined on the insert statement. In addition, all Quota sets under the existing account and/or user-id must be used.

Examples:

1. INSERT ACCOUNT AB23 MP=A DP=K DL=0 USERID XYZ; USING ACCOUNT 543

   QUIP will insert account AB23 with user-id XYZ. System standards will be used for those parameters that are not specified for the basic accounting block and for all parameters of the user-id entry. Account AB23 and user-id XYZ will share the Quota sets being used by account 543.

2. INSERT USERID TOP UQUOTA=6400
   UNDER ACCOUNT 321 USING ACCOUNT 998 USERID ABC

   QUIP will insert user-id TOP under account 321 using system standard values for those parameters that are not specified for the user-id entry. User-id TOP will share the Quota sets being used by user-id ABC under account 998.

3. INSERT USERID DEF UNDER ACCOUNT AAA USING USERID AAC

   QUIP will insert user-id DEF under account AAA using system standards for the user-id entry. User-id DEF will share the Quota sets being used by user-id AAC under account AAA.

QUIP prints the INSERT statement as it is read. After the entire statement is read, QUIP will process it and make the necessary calls on the file handling code to insert the accounts and/or user-ids into the summary account file. Then, if QUIP inserted an account, it will print a summary of the basic accounting block, a summary of all user-id entries and all Quota sets that were inserted. If a user-id was inserted, QUIP will print a summary of the user-id entry and all Quota sets that were inserted.

If any errors are encountered during any part of the insert process a self-explanatory message will be output.

8.3.2.2. READ Command

The READ command is used to read and display some part of the summary account file. This command has two formats. The first format is READ, with no other input parameters. This instructs QUIP to read and display all information in the summary account file, sorted by account.

The second format of the READ command is:

READ ACCOUNT *x* [USERID *y*] [SET *z*]

The second format of the READ command instructs QUIP to read and display information for a specific account in the summary account file. The optional fields may be used to limit the amount of information displayed to a specific user-id or even to a specific Quota set. Multiple accounts, user-ids or Quota sets may be specified.

If just the account field is specified, QUIP will display all of the information about the account. The information is displayed in the following order: basic accounting block, Quota sets for the account followed by every user-id entry under the account with the Quota sets for each user-id following the user-id entry to which they belong. This output format is also used by the first format of the READ command.

If the account and user-id fields are specified, QUIP will display the basic accounting block, the user-id entry for the user-id specified and the user-id's Quota sets.

If the Quota set field is specified, QUIP will display only the Quota set specified for the account or user-id specified.

Examples:

1. READ ACCOUNT ABC USERID STR,MNO

   QUIP will display the basic accounting block for account ABC followed by the user-id entry and Quota sets for user-id STR followed by the user-id entry and Quota sets for user-id MNO.

2. READ ACCOUNT BCD SET 4 SET 6 SET 3

   QUIP will display Quota sets 3, 4 and 6 for account BCD.

3. READ ACCOUNT DEF USERID XYZ SET 5

   QUIP will display Quota set 5 for user-id XYZ under account DEF.

Self-explanatory error messages will be displayed for any errors that are encountered while processing the READ command.

8.3.2.3. UPDATE Command

The UPDATE command is used to update any existing basic accounting block, user-id entry or Quota set. The format of the UPDATE command is:

UPDATE [ALL] ACCOUNT *x* [1] [USERID *y* [2]][SET *z* [3]]

where:

UPDATE	Command
ALL	Optional part of the command. It has meaning only when updating a Quota set. Because of the shared Quota set concept, the UPDATE command must specify if the update to a Quota set applies to just one account or user-id or to all accounts and user-ids that share the Quota set. ALL is used to specify this. If Quota sets are shared and ALL is not specified when updating one of the shared Quota sets, the account or user-id for which the update is being done is given its own copy of the shared Quota set and then the set is updated. After the update, the account or user-id for which the update was done has its own, unshared, Quota set. If the ALL is specified when updating a Quota set, it indicates to QUIP that the set should be updated for all accounts and user-ids that share the set. No additional copies of the Quota set are created and there is no change in the number of accounts and user-ids sharing the Quota set. If ALL is specified when Quota sets are not shared or when updating the basic accounting block or the user-id entry, it is ignored by QUIP.
ACCOUNT	Keyword
x	Account to update. Multiple accounts may be specified.
[1]	Optional field that specifies what parameters of the basic accounting block to update. Any of the parameters in Table 8-5 may be specified.

USERID — Optional keyword.

y — User–id to update. Multiple user–ids may be specified.

[2] — Optional field that specifies what parameters of the user–id entry to update. Any of the parameters in Table 8–5 may be specified. If a bit map in the user–id entry is being updated, all bits that are to be set at the termination of the update must be specified as input on the UPDATE command.

SET — Optional keyword.

z — Set number to update. If multiple sets are specified, each set number must be preceded by the keyword SET.

[3] — Optional field that specifies what parameters of the set to update. Any of the parameters in Table 8–6 may be specified. If a bit map in the Quota set is being updated, all bits that are to be set at the termination of the update must be specified as input on the UPDATE command. The number of units and time per unit must be specified when updating the T1 thru Tn and D1 thru Dn parameters.

Examples:

1. UPDATE ACCOUNT 123 MP=C

   QUIP will replace the contents of the MP cell of the basic accounting block of account 123 with C.

2. UPDATE ACCOUNT 234 SET 4 PAGES=2000 SET 6 PAGES=2000

   QUIP will replace the contents of the PAGES parameter of Quota sets 4 and 6 of account 234 with 2000. If the Quota sets used by account 234 are shared, account 234 will be given its own copy of the Quota sets before the update is performed.

3. UPDATE ALL ACCOUNT 456,789 USERID ABC UQUOTA=5000 SET 3 RRTL=24

   User–id ABC must exist under account 456 and under account 789. QUIP will replace the contents of the UQUOTA parameter of both user–id entries with 5000. In addition, QUIP will update the RRTL parameter of set 3 of both user–id entries to 24. If the Quota sets are shared, the update applies to all accounts and user–ids that share the sets. Additional copies of the sets are not created.

   Self–explanatory error messages will be displayed for any errors that are encountered while processing the UPDATE command.

8.3.2.4. ADD Command

The ADD command is used to add a Quota set to an existing account or user–id. The format of the ADD command is:

ADD [ALL] ACCOUNT *x* [USERID *y*] SET *z* [3]

where:

ADD — Command

ALL — Optional part of the command. ALL indicates that the set being added applies to all accounts and user-ids that share the existing sets. Absence of ALL indicates that the added set applies only to the account or user-id specified. If ALL is not specified and the existing sets are shared, QUIP gives the account or user-id specified on the ADD command its own copy of the shared Quota set and then performs the ADD. The account or user-id then has its own, unshared, copy of the Quota set.

ACCOUNT — Keyword

x — Account to use for the ADD. Multiple accounts may be specified.

USERID — Optional keyword.

y — User-id to use for the ADD. Multiple user-ids may be specified.

SET — Keyword

z — Number of the set to add (0-12). Multiple sets may be added. If multiple sets are added, the parameters for each set must be preceded by the keyword SET and the set number.

[3] — Optional field specifying any of the parameters in Table 8-7.

Examples:

1. ADD ACCOUNT 123 SET 1 START=0 STOP=600 LINES=700;
   SIZE=040 RMP=P SUPS=60 DES1=Q1/Q2/Q3 T1=2,40

   QUIP will add Quota set 1 to account 123. If the existing Quota sets used by account 123 are shared, QUIP will give account 123 its own copy of the existing Quota sets and then perform the ADD.

2. ADD ALL ACCOUNT 234 USERID BCD,DEF SET 4;
   START=0 STOP=400 PAGES=2000 SIZE=030 RMP=F;
   SUPS=60 DES1=Q1/Q3/Q9 T2=2,30 SET 2;
   START=1200.....

   QUIP will add Quota sets 2 and 4 to user-id BCD and to user-id DEF under account 234. If the existing Quota sets used by user-id BCD or DEF are shared, the added Quota sets will also be shared.

   Self-explanatory error messages will be displayed for errors that are encountered while processing the ADD command.

8.3.2.5. DELETE Command

The DELETE command is used to delete an account, user-id or Quota set. The format of the DELETE command is:

DELETE [ALL] ACCOUNT *x* [USERID *y*] [SET *z*]

where:

DELETE — Command

ALL — Optional part of the command. ALL has meaning only for deletion of a Quota set. If ALL is specified when deleting a shared Quota set, it indicates to QUIP that the set should be deleted for all accounts and user-ids that share the set. If ALL is not specified when deleting a shared Quota set, it indicates that the set should be deleted only for the account or user-id specified on the DELETE command. To do this, QUIP gives the account or user-id specified a copy of the shared sets and then performs the delete operation. The account or user-id then has its own unshared copy of the Quota set. If ALL is specified when deleting a Quota set that is not shared or when deleting an account or user-id, QUIP will ignore it.

ACCOUNT — Keyword

x — The account to delete if user-id and set are not specified. If user-id or set are specified, the deletion occurs under this account. Multiple accounts may be specified.

USERID — Optional keyword.

y — The user-id to delete if set is not specified. If set is specified the deletion occurs under this user-id. Multiple user-ids may be specified.

SET — Optional keyword.

z — Set number to delete. If multiple Quota sets are specified, each Quota set must be preceded by the keyword SET.

Examples:

1. DELETE ACCOUNT 678

   QUIP will delete account 678 along with all associated user-ids and Quota sets if the sets are not shared.

2. DELETE ACCOUNT 123,345 USERID CBC,DEF

   QUIP will delete user-ids CBC and DEF from account 123 and from account 345 along with the Quota sets for the user-ids if the sets are not shared. If the Quota sets are shared by other accounts and/or user-ids, the Quota sets are not deleted.

3. DELETE ACCOUNT 234 SET 7 SET 8

   QUIP will delete Quota sets 7 and 8 from account 234. If the Quota sets used by account 234 are shared, QUIP will give account 234 a copy of the shared sets and then delete them.

4. DELETE ALL ACCOUNT 456 SET 8

   QUIP will delete Quota set 8 from account 456. If the Quota set is shared, the DELETE applies to all accounts and user-ids that share the set.

After processing the DELETE command, QUIP outputs a list of accounts, user-ids and Quota sets that were deleted.

Self-explanatory messages are printed for any errors that are encountered while processing the DELETE command.

8.3.2.6. PURGE Command

The PURGE command is used to reset the accumulators for an account or user-id to zero. The PURGE command has two formats. These formats are:

PURGE

PURGE ACCOUNT *x* [USERID *y*]

The PURGE command with no fields specified will reset the accumulators for all accounts and user-id entries in the summary account file. If an account only is specified on the PURGE command, QUIP resets all accumulators for the account and all user-ids under the account. If an account and user-id are specified on the PURGE command, QUIP resets the accumulators only for the user-id. Multiple accounts and user-ids may be specified.

Examples:

1. PURGE ACCOUNT 123,567

   QUIP will purge all accumulators for accounts 123 and 567 and also purge all user-ids under both accounts.

2. PURGE ACCOUNT 345 USERID ABC,DEF,GHI

   QUIP will purge the accumulators for user-ids ABC, DEF and GHI under account 345.

After processing the PURGE command, QUIP outputs a list of accounts or user-ids that were purged.

Self-explanatory error messages are printed for error cases encountered while processing the PURGE command.

8.3.2.7. LIST Command

The LIST command is used to list all account names that are present in the summary account file or to list all user-id names under an account. The two formats of the LIST command are:

LIST ACCOUNTS [AND USERIDS]

LIST USERIDS UNDER ACCOUNT *x*

The first format lists all account names in the summary account file and, if the optional USERIDS field is specified, lists all user-id names under every account. Multiple accounts may be used for format two.

Self-explanatory error messages are output for errors encountered while processing the LIST command.

NOTE:

The Executive uses a third of a word to pass the number of accounts in the Summary Account File to the QUIP processor. If there are more than 2047 accounts present in the file, the cell containing the number of accounts would have an indeterminate value.

8.3.2.8. CHANGE Command

The CHANGE command is used to change the master account and/or user-id. The format of the CHANGE command is:

CHANGE MASTER ACCOUNT *x* [USERID *y*]

where:

CHANGE	Command
MASTER ACCOUNT	Keywords
x	New master account. This account must exist before performing the change.
USERID	Optional keyword.
y	New master user-id. This user-id must exist before performing the change.

Example:

CHANGE MASTER ACCOUNT 98765 USERID CDEFG

QUIP will change the master account to 98765 and the master user-id to CDEFG. After this command is processed, the run is no longer privileged because the master account has changed.

QUIP outputs a message indicating that the master account and user-id are changed. Self-explanatory error messages are printed for any errors that are encountered while processing the CHANGE command.

8.3.2.9. END Command

The END command is an optional end-of-source indicator.

8.3.3. File Handling Code

Under the Quota System, the summary account file cannot be directly accessed by a user. The ER ACCNT$ provides an interface between the user and the summary account file. The code is turned on for Quota levels one thru four and it is completely independent of the monitoring and enforcing code.

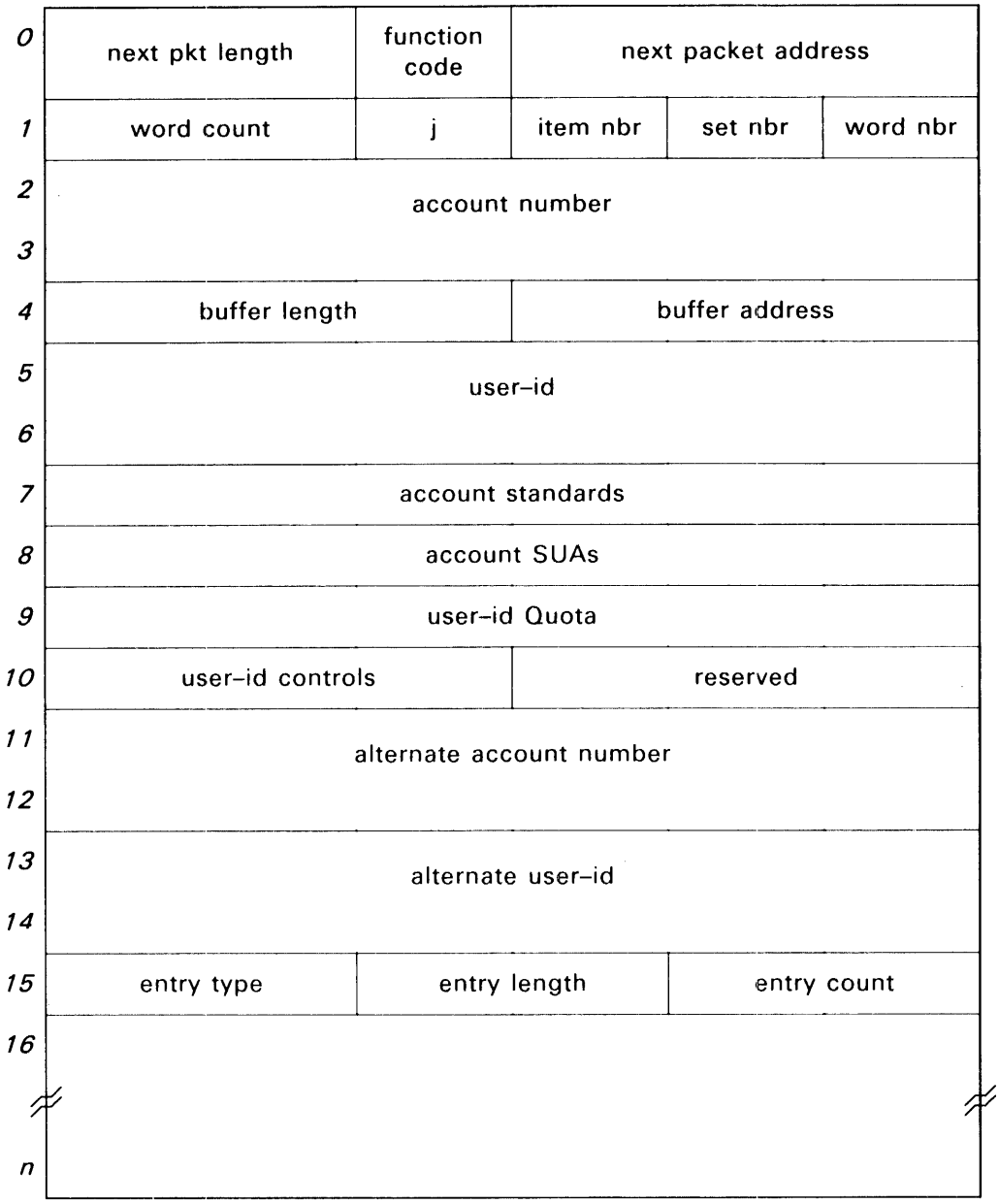
Operations on the summary account file are initiated by the following sequence of instructions.

```
L     A0,(packet length, packet address)
ER    ACCNT$
```

Since the ER packet length is variable, the packet length supplied when doing the ER call is the number of words being used for this particular ER call.

The format of the packet is shown in Table 8-8 and described below.

Table 8–8. ER ACCNT$ Packet



NOTE:

Words 0 through 14 are the regular packet and words 15 through n are the packet extension. Use of the packet extension depends on the function code used (Word 0, S3).

Word 0

next packet length — This field specifies the total word length of the next packet as explained under next packet address.

function code — The allowable function codes are:

Code	Function
020	INSERT
001	RETRIEVE
030	UNLINK
031	RELINK
022	UPDATE
021	ADD
011	DELELTE
010	PURGE
002	LIST
040	CHANGE

next packet address — This field must be zero unless chaining of several packets is desired on a particular ER ACCNT$ call. For example, packet chaining might be used to retrieve or update selected words from the accounting block, user-id entry, and Quota set, all in a single ER ACCNT$ call. Packets containing different function codes can also be chained. The last packet in the chain is identified by a zero in the next packet address field. A maximum of 10 packets may be chained together per ER.

Word 1

word count — This field has no meaning for input on the ER call. For update and retrieve functions, SACCNT stores the number of words updated or retrieved in this field.

j — This field is used for the update function only. When it is specified, one partial word is updated. The values for j used by SACCNT are equivalent to the assembler language j values.

item number — This field specifies the item being referenced. The description of the separate ER ACCNT$ functions specifies when the item number is used. The item number values with their meanings are:

0 – accounting block
1 – user-id entry
2 – Quota summary sector
3 – Quota sets

set (or sector) number — This field identifies the Quota set to update or delete. For the retrieve function, if the item number is three, it identifies the Quota set with which to begin the retrieval. It is also used to designate the second half of the basic accounting block for update and retrieval.

word number — This field is used only for the update or retrieve function. It identifies the word with which to begin the update or retrieval.

Words 2,3

account nbr — This 12-character field identifies the account upon which the operation will be performed.

Word 4

buffer length and buffer addr — These fields specify the length and address of a buffer which contains additional information not found in the packet. The description of the separate ER ACCNT$ functions indicates where buffers are required.

Words 5,6

user-id — This 12-character field identifies the user-id upon which the operation will be performed.

Word 7

account standards — This field is used only for the insertion of an account. It is used to specify the account standards contained in word 5 of the basic accounting block (see 8.2.4, TSS File Structure). This field is divided into sixths with each sixth being used as follows:

S1 - MP
S2 - DP
S3 - DL
S4 - RTL
S6 - CEQ
} Parameter-ids are defined in Table 8-5.

S5 - 0 — used internally by the EXEC

If a value is not specified for any given sixth (i.e., zero specified), SACCNT supplies system standards for that sixth.

However, if a user wants the sixth to remain zero, a value of 077 is placed in that sixth of the packet. This is then converted to zero when building the basic accounting block. The use of 077 in this manner precludes the use of it as a legitimate input value. An exception to the 077 convention is made for parameters MP and DP. If 077 is specified for them, they are given the system standard values.

Word 8

account SUAs — This field is used for the insertion of an account only. It is used to specify the SUA Quota for the account. If it is set to zero, the system standard will be received. A minus zero in the packet indicates that the field should be zero in the basic accounting block.

Word 9

user-id Quota — This field is used only when inserting a user-id. It is used to insert the SUA Quota for the user-id. If the field is set to zero, the user-id receives the system standard user-id quota. Minus zero in this field indicates that the user-id Quota in the user-id entry should be set to zero.

Word 10

user-id controls — This field is used only when inserting a user-id. It is used to insert a bit map describing what operations the user-id is allowed to perform. If zero is specified, zero is put in the user-id entry. Bits 0 and 1 are used internally by the EXEC. Bits 4 and 5 are used to designate a user-id as a submaster as described in 8.3.2.1. The remaining bits are unassigned.

Words 11 thru 14

alternate account and user-id — These fields are used for the insert and relink functions only. The fields must be zero unless it is desired to use all of the existing Quota sets defined for another account and/or user-id when inserting a new account and/or user-id or relinking an account and/or user-id.

Words 15 thru n

This is the packet extension and is used only for the insertion of an account. The purpose of the packet extension is to allow new or expanded information to be input without changing the basic format of the packet. Each group of entries in the packet extension must be preceded by an extension control word. The extension control word consists of three parts. They are:

entry type — Identifies what kind of information is contained in the entries that follow. At present, only type zero is defined. Type zero consists of single word entries defining the catalogued mass storage quotas for an account. Each mass storage Quota entry must be right justified and entries must appear in the same order as they appear in the basic accounting block. In addition, starting with group one in the basic accounting block (see 8.3.4), there must be a corresponding entry in the packet extension for every catalogued mass storage Quota up to and including the last one in the group that the user chooses to give an initial value. For example, if the user wants to give an initial value for mass storage group three, give values for groups one and two must first be given. For entries not specified, (this could be all entries), system standards are used. If a value of zero is specified for a catalogued mass storage Quota in the packet extension, system standards are used. If a negative value is specified in the packet extension, zero is used as the Quota value.

entry length — Defines the length of each entry under this control word. Type zero entries are always one word long.

entry count — Defines the number of entries under this control word.

Any fields of the ER packet that are not used, but are included in the packet length, must be set to zero. On return from the ER ACCNT$, A0 is set positive to signify normal completion and negative if the operation was unsuccessful. If negative, H2 of A0 contains the address of the packet in error and S2 contains one of the error codes listed in Table 8-9.

Table 8-9. Error Codes

Code	Description
001	Invalid function code
002	Packet not within limits
003	Buffer not within limits
004	Account does not exist
005	User-id does not exist
006	Alternate account does not exist
007	Alternate user-id does not exist
010	Account already exists (function 020)
011	User-id already exists (function 020)
012	I/O error (actual I/O status in S3)
013	Buffer too small to contain the complete retrieval to the end of the item chain. This is a warning only and A0 is not set negative. If chained packets are used, the warning is given only for the last packet in the chain.
014	Item number not valid
015	Sector number not valid
016	Word number not valid
017	Invalid j designator
020	Zero account number not allowed
021	Invalid packet length
022	Quota set exists (set number in S3)
023	Time overlaps in Quota sets (offending set number in S3)
024	Packet chaining maximum exceeded
025	Delete not allowed
026	Conflict in Quota set definition
027	Invalid entry count in extension control word
030	Buffer too large for update of item

Table 8-9. Error Codes (continued)

Code	Description
031	No Quota sets given on add or relink function
032	PCT abort due to illegal operation attempt
033	Attempt to assign illegal privileges
034	Format of specified Quota set does not agree with AACONFIG definition

Only privileged runs are allowed to use all of the functions of ER ACCNT$. Non-privileged runs are only allowed to retrieve information concerning their own account and user-id.

The following describes what is accomplished by each ER ACCNT$ function and how to use each function. See 8.3.4 for an explanation of the summary account file structure.

8.3.3.1. INSERT

The INSERT function is used to insert accounts and/or user-ids into the summary account file. The item field is set to zero to insert an account and it is set to one to insert a user-id under an existing account. All input parameters for the user-id entry and the basic accounting block are specified in the ER packet. Every account and user-id must have at least one Quota set. If an account is inserted without a Quota set specified, the account shares the system standard Quota sets. A user-id inserted with an account shares the Quota sets of the account. User-ids inserted under an existing account without specifying a Quota set share the sets of the account under which they are inserted. If Quota sets are specified when an account or user-id is inserted, the contents of the Quota sets are specified in a buffer whose address is placed in the buffer address field of the ER packet. Multiple Quota sets may be specified in a buffer. Each Quota set must be 28 words in length and must follow the format of a Quota set as set forth in 8.3.4. SACCNT moves the Quota set, exactly as it appears in the buffer, to the summary account file. It is the responsibility of the user to supply all necessary information for the Quota set in the input buffer. The input buffer length must be a multiple of 28.

If an account and/or user-id is inserted using the Quota sets of an existing account and/or user-id, the name of the account and/or user-id whose Quota sets are used is placed in the appropriate alternate account and/or alternate user-id cells of the ER packet. When an account is inserted in this manner, the alternate account must always be specified. When a user-id is inserted in this manner, and an alternate user-id is specified, the alternate account must be specified only if it differs from the specified account.

8.3.3.2. RETRIEVE

The RETRIEVE function is used to retrieve information from the summary account file about an account, user-id or Quota set. The hierarchy of items for a retrieval is basic accounting block, user-id entries, summary sector, and Quota sets. The retrieved information is placed in the buffer supplied by the user. The retrieval begins at the item, set (or sector) and word specified in the ER packet, and continues until either the buffer is full or until all items have been retrieved. If the buffer is filled before all items are retrieved, a status of 013 is set in S2 of A0 as a warning that all the available information was not retrieved, but A0 is not set negative. Note that if linked packets are used, the 013 status is given only for the last packet in the chain. Therefore, if the user wants the 013 status for every retrieval that does not receive all of the available information, linked packets should not be used.

If a user-id is specified in the ER packet, the retrieved information is about the specified user-id. If the user-id field is not specified, the information retrieved is about the account. The format of the retrieved information when a user-id is specified is 56 words of basic accounting block for the account under which the user-id resides, a 5-word user-id entry for the specified user-id, the user-id summary sector and a 28-word entry for every user-id Quota set. The format of the retrieved information when a user-id is not specified is 56 words of basic accounting block followed by all user-id entries under the account. The user-id entries are in 28-word blocks followed by a 28-word entry for each Quota set of the account. Quota sets are always retrieved in numerical order starting with zero, if it exists. All retrieved items have the format set forth in 8.3.4. The word count field of the ER packet reflects the number of words retrieved.

This is the only ER ACCNT$ function that a non-privileged run is allowed to perform. A non-privileged run can retrieve information only about its own account, user-id and Quota sets. A submaster is allowed to retrieve information about its own account and all user-ids under the account.

8.3.3.3. UNLINK

The UNLINK function allows the user to break off an account or user-id from a shared group of Quota sets and duplicate these Quota sets elsewhere in the summary account file so they are used only by the specified account or user-id. The presence of an entry in the user-id field of the ER packet indicates that the user-id is to be unlinked rather than the account. No buffer is used for this function. If the account or user-id is already unlinked, the Executive returns to the caller with a normal status.

8.3.3.4. RELINK

The RELINK function allows the user to redefine the Quota sets for an account or user-id. This is done by breaking off the account or user-id from the Quota sets that it is currently using and attaching it to a previously defined group of Quota sets. The account and user-id fields of the ER packet specify the account or user-id to be relinked. If a user-id is specified along with the account, only the user-id is relinked. If no user-id is specified, the account is relinked. The alternate account and alternate user-id fields of the ER packet specify the name of the account or user-id whose Quota sets are used for the relink. The relinked account or user-id then shares the Quota sets with the alternate account or alternate user-id. A buffer is not used for this function.

8.3.3.5. UPDATE

The UPDATE function is used to update a basic accounting block, a user-id entry or a Quota set. Words to be updated are always passed in the buffer, not in the packet. The item, word and set fields of the ER packet indicates where the update is to begin. The update continues until the end of the buffer is reached. Only one item or set may be updated per ER packet. If the buffer is larger than the remainder of the item or set being updated, the update is not begun and an error status is returned. For example, if the last five words of a Quota set are to be updated, but the buffer length is six, the update is not begun and an error status is returned.

The format of the buffer is exactly the same as the format of the words being updated (see 8.3.4). A partial word of an item or set can be updated by setting the j field of the ER packet to indicate what part of the word to update. When doing a partial word update, the buffer length must be one and the update value is right justified in the single word buffer.

The Quota summary sector cannot be updated directly, but is automatically updated by modifying a Quota set.

If a shared Quota set is updated, the update applies to all accounts and/or user-ids that share the set. If the update should apply to one or a few of the accounts and/or user-ids that share the set, those accounts and/or user-ids must be unlinked before doing the update.

Words zero thru four of the basic accounting block and words zero and one of the user-id entry cannot be updated. Any attempt to update these words results in an error status. Therefore, when updating the basic accounting block, the first word that may be updated is word five. If the update is started in the first 28 words of the basic accounting block, set (sector) is set to zero and the second 28 words may also be updated at the same time. However, if the update starts in the second 28 words of the basic accounting block, set (sector) must be set to one.

8.3.3.6. ADD

The ADD function is used to add additional Quota sets to an existing account or user-id. The Quota sets are defined in the buffer and have the format of a Quota set described in 8.3.4. Multiple Quota sets may be added. The buffer length must be a value that is a multiple of 28. Any information that the user wants in the Quota set must be supplied by the user. The information supplied by the user is never altered.

If the account or user-id to which the sets are being added shares Quota sets, the added Quota sets apply to all user-ids and accounts that share the sets. If the added sets should apply to a single account or user-id, the account or user-id must be unlinked before doing the ADD.

If multiple sets are added and one of the sets being added already exists, an error status is returned and none of the sets are added.

8.3.3.7. DELETE

The DELETE function is used to delete an account, a user-id or a Quota set. When an account is deleted, all user-ids and Quota sets, if they are not shared by another account, are deleted. When a user-id is deleted, all Quota sets for the user-id are also deleted, if they are not shared. If a Quota set is deleted from a group of shared Quota sets, the set is deleted from all accounts and user-ids that share the set. If a shared set should be deleted for a single account or user-id, the account or user-id must be unlinked before doing the delete of the set.

Deletion of a Quota set is identified by setting the item field in the ER packet to three. If the item field is not three and a user-id is specified in the user-id field of the ER packet, the user-id is deleted. An account is deleted if there is no user-id specified and the item field is not equal to three. No buffer is used for this function.

A use count is maintained for each group of Quota sets. When an account or user-id is deleted, the use count is decremented, but the Quota sets are not released until the use count becomes zero.

8.3.3.8. PURGE

The PURGE function is used to clear the accumulators for a particular user-id or for an entire account. When the account is purged, all user-ids under the account are automatically purged. To purge an entire account, specify only the account name in the ER packet. If an account and user-id are specified, only the user-id is purged.

When purging an account, the purge time is recorded in the basic accounting block and first entry time is cleared to zero.

8.3.3.9. LIST

The LIST function is used to obtain a list of all accounts in the summary account file. The name of every account is entered as a 2-word entry in the buffer specified in the ER packet. The number of words put in the buffer is entered in the word count field before returning to the caller. A status of 013 returned in this case warns the user that the buffer is too small to contain the entire list of accounts. The account field of the ER packet is set to zero for this function.

8.3.3.10. CHANGE

The CHANGE function is used to change the master account and user-id. The new master account and user-id are specified in the account and user-id fields of the ER packet and they must exist in the summary account file before doing the change. The old master account and user-id are not deleted by this function. Note that the run doing the change is not a privileged run after the change is completed, because it is no longer running under the master account and user-id.

8.3.4. File Structure

The summary account file consists of a header (sector zero), a lookup table (variable length) and entries for all accounts in the system. The account entries are: basic accounting block, user-id entries, Quota summary sector and Quota sets. The format of each entry is presented below. Figure 8-2 presents a general layout of the file showing the relationship between the entries in the file.

8.3.4.1. Summary Account File Header

Sector zero of the summary account file is referred to as the header and is used by SACCNT to keep general information. The header is automatically initialized at boot time and is updated by SACCNT when necessary. Only 12 words of the header are used. The format of the header is shown in Table 8-10 and described below.

Table 8-10. Summary Account File Header

0	ATOCL	
1	one-sector chain	two-sector chain
2	four-sector chain	EOF
3	system standard account basic account block sector address	system standard account
4	account header	
5		

Table 8-10. Summary Account File Header (continued)

6 7	QUOTA header
8 9	MASTER account
10 11	MASTER user-id

Word 0

ATOCL — Is the length of the lookup table (see 8.3.4.2).

Word 1

one-sector chain — Address of the first available one sector slot or 0.

two-sector chain — Address of the first available two sector slot or 0.

Word 2

four-sector chain — Address of the first available four sector slot or 0.

All entries in the summary account file are either one, two or four sector entries. When an entry is deleted from the summary account file, the slot where the entry was is now available to be re-used for another entry. Open slot chains are maintained for these previously allocated but currently unused sectors. The first entry for each chain is kept in the fields described above. The area listed in the open slot chains will be reused before allocating additional file space.

When additional file space is allocated, the number of sectors allocated at one time is specified by the SGS RECSIZ. If the summary account file resides on disk, RECSIZ should equal the number of sectors per record. Then one record at a time will always be allocated allowing multiple sector entries to reside in the same record. If the summary account file is not on disk, set RECSIZ to four. If more sectors than are currently needed are allocated, the extra sectors are placed in the open slot chains.

EOF — Address of the first previously unallocated sector in the summary account file.

Words 3 through 11

sys std acct summary sector — Is the sector address of the Quota summary sector for the system standard account.

sys std acct acct blk sector address — Is the sector address of the Quota basic accounting block for the system standard account.

account header — Is the header for the basic accounting block that is being used by this system.

QUOTA header — Is the header for the Quota set that is being used by the system.

MASTER account — Is the name of the current master account.

MASTER user-id — Is the name of the current master user-id.

8.3.4.2. Lookup Table

To minimize the amount of time required to locate an account in the summary account file, a lookup table is provided. The lookup table consists of N consecutive words starting in sector one, where N is the SGS parameter ATOCL and a prime number. The lookup table index generation hashing function consists of adding the two 6-character halves of the account name, dividing the result by the lookup table length, and using the remainder as the index into the lookup table. Entries in the lookup table are 1-word entries which are either zero or, if positive, an index to a basic accounting block or, if negative, an index to a single-sector overflow page which will contain up to nine 3-word account name entries. Each account name entry consists of the account name and the address of the basic accounting block. Multiple overflow pages can exist for each index in the lookup table. Each page is linked forward from the previous page and a zero link address terminates the scan of the overflow pages.

8.3.4.3. Basic Accounting Block

The basic accounting block is shown in Table 8-11.

Table 8-11. Basic Accounting Block

0	1	0	0	*reserved*		*Quota summary link*	
1	*account number*						
2							
3	*ACCS*	*ACCL*	*CMSS*	*CMSL*	*TMPSS*	*TMPSL*	
4	*TUSS*	*TUSL*	*DUSS*	*DUSL*	*SYMS*	*SYML*	
5	*max priority*	*default priority*	*deadline*	*real time level*	*operator add entry*	*can exceed Quota*	
6	*maximum SUAs*						
7	*SUAs to date*						
8	*first entry time*						
9	*last entry time*						

Table 8-11. Basic Accounting Block (continued)

10	entry last cleared	
11	reserved	user-id link
12	SUPs to date	
13	CBSUPs to date	
14	lines out to date	
15	cards in to date	
16	cards out to date	
17	voluntary delay time to date	
18	batch runs to date	demand runs to date
19	number of catalogued files	reserved
20	max. catalogued tracks, 1	catalogued tracks to date, 1
	≈	≈ ≈
n	max. catalogued tracks, n	catalogued tracks to date, n
	track-SUPs-in-seconds on type 1	
	≈	≈
	track-SUPs-in-seconds on type n	
	tape-unit-SUPs-in-seconds on type 1	
	≈	≈
	tape-unit-SUPs-in-seconds on type n	
	disk-unit-SUPs-in-seconds on type 1	
	≈	≈
	disk-unit-SUPs-in-seconds on type n	
	symbiont-unit-SUPs-in-seconds type 1	
	≈	≈
52	symbiont-unit-SUPs-in-seconds type n	

Table 8-11. Basic Accounting Block (continued)

53	*reserved*
54	*reserved*
55	*reserved*

Word 0

Quota summary link — Is the sector address of the Quota summary sector for this account.

Words 1 and 2

account number — Is the number of this account.

Words 3 through n

ACCS — Is the start of the fixed section of the basic accounting block relative to the start of the basic accounting block.

ACCL — Is the length of the fixed section.

CMSS — Is the start of the catalogued mass storage section of the basic accounting block relative to the start of the basic accounting block. This section normally follows immediately after the fixed section.

CMSL — Is the length of the catalogued mass storage section.

TMPSS — Is the start of the temporary mass storage section of the basic accounting block relative to the start of the basic accounting block. This section normally follows immediately after the catalogued mass storage section.

TMPSL — Is the length of the temporary mass storage section.

TUSS — Is the start of the tape unit section of the basic accounting block relative to the start of the basic accounting block. This section normally follows immediately after the temporary mass storage section.

TUSL — Is the length of the tape unit section.

DUSS — Is the start of the disk unit section of the basic accounting block relative to the start of the basic accounting block. This section normally follows immediately after the tape unit section.

DUSL — Is the length of the disk unit section.

SYMS — Is the start of the symbiont section of the basic accounting block relative to the start of the basic accounting block. This section normally follows immediately after the disk unit section.

SYML — Is the length of the symbiont section.

Words three and four of the basic accounting block are referred to as header words. The header word values are defined in AACONFIG, and they can not be changed after the summary account file is initialized. If a recovery boot is done with a system whose header words are different than those currently in use in the summary account file, the summary account file must be reinitialized.

max priority — Is the maximum priority allowed for this account {A–Z}.

default priority — Is the priority used when none is specified on the RUN image.

deadline — If set to one, deadline runs are allowed for this account. Zero indicates that deadline runs are not allowed.

real time level — A nonzero value indicates that real–time runs are allowed and also indicates the highest level allowed. A zero indicates that real–time runs are not allowed.

operator add entry — A bit map used internally by the EXEC.

Bit	
0	account was entered by an answer of 'A' to the undefined account message.
1	account was entered by an answer of 'E' to the undefined account message.
2–5	unused.

can exceed Quota — If set to zero, the account cannot exceed its Quota limit. If set to one, the account is allowed to exceed its Quota limit. This applies only to SUAs.

max SUAs — Are the maximum SUAs the account is allowed to use.

SUAs to date — Are the number of SUAs that runs executing under this account have accumulated thus far.

first entry time — Is the time (TDATE$ format) when the EXEC accounting routine first updated this accounting block.

last entry time — Is the time (TDATE$ format) when the EXEC accounting routine last updated this accounting block.

entry last cleared — Is the time (TDATE$ format) of the last purge of the accounting block.

user–id link — Is the sector address of the first group of user–id entries.

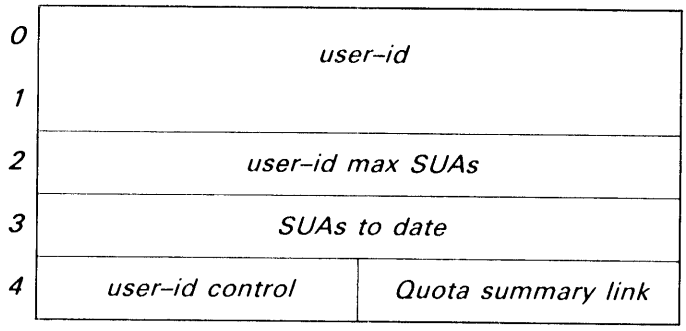
SUPs to date — Is the total number of SUPs, expressed in seconds, accumulated thus far for this account.

CBSUPs to date — Is the total number of core block SUPs, expressed in core block seconds, accumulated thus far for this account.

lines out to date	Is the number of lines printed for this account.
cards in to date	Is the number of cards read for this account.
cards out to date	Is the number of cards punched for this account.
voluntary delay time to date	Is the total time, expressed in seconds, that runs under this account were in a voluntary wait state.
batch runs to date	Is the number of batch runs processed under this account.
demand runs to date	Is the number of demand runs processed under this account.
number cat. files	Is the number of files that have been catalogued by runs using this account.
max cat. tracks 1 thru n	Specifies, by mass storage group, the number of tracks the account is allowed to use for catalogued files. The groups are defined in AACONFIG. The number of groups (n) is specified by cell CMSL of the header.
cat. tracks to date 1 thru n	Is the number of tracks, per mass storage group, that the account has used for catalogued files.
track SUPs in seconds on type 1 thru n	Is the total number of temporary track-SUPs, expressed in seconds, per fixed mass storage group, that the account has accumulated. The number of fixed mass storage groups (n) is specified by cell TMPSL of the header.
tape unit SUPs in seconds on type 1 thru n	Is the total number of tape-unit-SUPs, expressed in seconds, per tape group, that the account has accumulated. The number of tape groups (n) is specified by cell TUSL of the header. Tape groups are defined in AACONFIG.
disk unit SUPs In seconds on type 1 thru n	Is the total number of disk-units-SUPs, expressed in seconds, per disk group, that the account has accumulated. The number of disk groups (n) is specified by DUSL of the header. Disk groups are defined in AACONFIG.
symbiont unit SUPs in seconds type 1 thru n	Is the total number of symbiont-unit-SUPs, expressed in seconds, per symbiont group, that the account has accumulated. The number of symbiont groups (n) is specified by SYML of the header. Symbiont groups are defined in AACONFIG.

8.3.4.4. User–id Entry

Every user–id has a five word entry that describes the user–id. The format of this entry is:



where:

user–id — Is the name of the user–id.

user–id max SUAs — Is the maximum number of SUAs that this user is allowed to use.

SUAs to date — Is the number of SUAs that this user has used thus far.

user–id control — Is a bit map used to describe and control this user–id.

Bit

0 — User–id was entered by an answer of 'A' to an undefined user–id message.
1 — User–id was entered by an answer of 'E' to an undefined user–id message.
4 — Can manipulate Quota sets if the user–id is a submaster.
5 — This user–id is a submaster.

The remaining bits are unused.

Quota summary link — Is the sector address of the Quota summary sector for this user–id.

These 5–word entries are blocked together into groups of five entries and placed in a 28–word sector of the summary account file. The sector address of the first block of user–id entries is placed in the user–id link of the basic accounting block. H2 of word zero of each user–id sector contains the sector address of the next block of user–id entries. A zero forward link indicates the end of the chain. The user–id entries begin at word one of each sector.

8.3.4.5. Quota Summary Sector

The Quota summary sector contains a 2-word entry for each of the possible 13 Quota sets that an account or user-id can have. It is automatically updated every time a Quota set is inserted, updated or deleted. The format of the Quota summary sector is shown in Table 8-12 and described below.

Table 8-12. Quota Summary Sector

0	1	0	1	*reserved*		*use count*
1	*reserved*					
2	*Quota start*					*Quota stop*
3	*DES1 bits*		*reserved*		*priority*	*link to set 0*
4	*Quota start*					*Quota stop*
5	*DES1 bits*		*reserved*		*priority*	*link to set 1*
6						
≈						≈
25						
26	*Quota start*					*Quota stop*
27	*DES1 bits*		*reserved*		*priority*	*link to set 13*

Word 0

use count	Is the number of accounts and user-ids that are using this Quota summary sector.

Words 2 through 27

Quota start	Is the start time, in minutes past midnight, specified in the Quota set.
Quota stop	Is the stop time, in minutes past midnight, specified in the Quota set.
DES1 bits	Bits 35, 34, 33, and 32 of descriptor word one are duplicated here. They are used for run scheduling.
priority	Is the priority specified in the Quota set.
link to set	Is the sector address of the Quota set.

8.3.4.6. Quota Set

The Quota set is shown in Table 8–13 and described below.

Table 8–13. Quota Set

0	0 0 0	Quota Start Time		Quota Stop Time		
1	FIXS	FIXL	TAPS	TAPL	DISCS	DISCL
2	MASS	MASL	Reserved		Real Time Level	Quota Set nbr
3	pages out	cards out				
4	Peak Core Size		Priority	Connect Time		
5	SUPs					
6	Voluntary Delay Time (VDT)					
7	Descriptor 1					
8	Descriptor 2					
9	reserved					
10	Number Of Units, Tape 1			Time/Unit, SUPs + VDT		
	Number of Units, Tape n			Time/Unit, SUPs + VDT		
n	Number Of Units, Disk 1			Time/Unit, SUPs + VDT		
	Number of Units, Disk n			Time/Unit, SUPs + VDT		
n	Temporary Tracks, Type 1			Catalogued Tracks, Type 1		
27	Temporary Tracks, Type n			Catalogued Tracks, Type n		

where:

Quota Start Time — Specifies the beginning of the time interval to which the limits in the Quota set apply. Time is specified in minutes past midnight using a 24–hour clock.

Quota Stop Time — Specifies the end of the time interval to which the limits in the Quota set apply. Time is specified in minutes past midnight using a 24–hour clock.

FIXS	Specifies the start of the fixed section of the Quota set relative to the start of the Quota set.
FIXL	Specifies the length of the fixed section.
TAPS	Specifies the start of the tape unit section of the Quota set relative to the start of the Quota set.
TAPL	Specifies the length of the tape section.
DISCS	Specifies the start of the disk unit section of the Quota set relative to the start of the Quota set.
DISCL	Specifies the length of the disk section.
MASS	Specifies the start of the mass storage section of the Quota set relative to the start of the Quota set.
MASL	Specifies the length of the mass storage section.

NOTE:

Word one and T1 of word two of the Quota set are referred to as the Quota set header. These values are defined by SGS and they must also be supplied by the user in every Quota set that is inserted or added. The set is not built, but rather it is merely copied from the user buffer to the summary account file. The values for the header words cannot be changed after the summary account file is initialized. If a recovery boot is done with a system whose header words are different than those currently in use in the Quota sets, the summary account file must be reinitialized (see 8.3.5.3).

Real Time Level	Specifies the real time level at which a run using this Quota set is allowed to execute. The level given in this Quota set overrides the level given in the basic accounting block.
Quota Set Number	Specifies the number used to identify this Quota set under a given account or user-id. Every Quota set under an account or user-id must have a unique set number in the range from 0 to 12.
Pages Out	The maximum number of pages that a run is allowed to output.
Cards Out	The maximum number of cards that a run is allowed to output.
Peak Core Size	Defines the maximum program size, in blocks, that will be loaded for a run or allowed for MCORE$ expansion.
Priority	The maximum priority allowed for the run (A thru Z). The priority in the Quota set overrides the maximum priority in the basic accounting block.
Connect Time	This parameter applies to demand runs only and specifies the maximum amount of time that a run can use a terminal. Time is specified in seconds.
SUPs	Defines the maximum number of SUPs, expressed in seconds, that a run is allowed to accumulate. This parameter cannot be overriden by the maximum time parameter on the RUN image.
Voluntary Delay Time	Specifies the maximum voluntary delay time, in seconds, that a run is allowed to accumulate.

Descriptor 1 — A bit map used to define various capabilities, limitations and attributes of the run using the Quota set.

Bit	
35	quotas apply to onsite batch runs.
34	quotas apply to remote batch runs
33	quotas apply to demand runs.
32	deadline time allowed.
31	use of 'V' and 'G' options on @ASG is allowed.
30	absolute device name assign, arbitrary device assign and assignment of a communications device are allowed.
29	Checkpoint/Restart is allowed. Checkpoint/Restart console keyin is always allowed.)
28	ER RSI$ allowed
27	SIP data reduction allowed
26	SIP ON/OFF allowed via ER SYSBAL$
25	ADH access to labeled packs is allowed
24	All SIP functions allowed via ER SYSBAL$

Descriptor 2 — Similar to descriptor 1. Bit meanings have not been defined.

Number of Units, Tape 1–n — Specifies the number of tape units, per tape group, that a run can have assigned at one time. The number of tape groups (n) is specified by TAPL in the Quota set header words.

Time/Unit — Specifies, in seconds, the length of time each unit in the groups can be assigned.

Number of Units, Disk 1–n — Specifies the number of disk units, per disk group, that a run can have assigned at one time. The number of disk groups (n) is specified by DISCL in the Quota set header words.

Time/Unit — Specifies, in seconds, the length of time each disk unit in the group can be assigned.

Temporary Tracks Type 1 thru n — Specifies the maximum number of tracks of temporary mass storage, per mass storage group, that a run is allowed to use. The number of mass storage groups (n) is specified by MASL in the Quota set header words.

Catalogued Tracks Type 1 thru n — Specifies the maximum number of tracks of catalogued mass storage, per mass storage group, that a run is allowed to use. The number of mass storage groups (n) is specified by MASL in the Quota set header word.

The types of equipment in each of the above groups are defined by SGS at system generation.

The start and stop times of Quota sets for the same run type cannot overlap. Start and stop times of zero indicate that no time exists for the Quota set. If the start and stop times of a Quota set are equal, and not zero, the Quota set applies around the clock.

8.3.4.7. File Layout

See Figure 8–2 for file layout of the summary account file.

8.3.5. Configuration and System Generation

This subsection describes the configurable parameters in the Quota system and explains how they are used.

8.3.5.1. Possible System Configurations

Three SGS parameters combine to dictate the Quota system configuration. The parameters are:

ACCTON defines the Quota level (0–4)

RESTRICT specifies if the operator should be solicited for undefined accounts and/or user–ids.

USERON specifies if user–id checking and validation is turned on in the system.

These three parameters combine to make a possible 18 different configurations. They are:

- Configuration 1

  Quota level zero, user–ids off. RESTRICT has no meaning in this system. The summary account file does not exist and all accounts are allowed to run.

- Configuration 2

  Quota level zero, user–ids on. RESTRICT has no meaning in this system. The summary account file does not exist and all accounts are allowed to run. However, every run must specify a user–id. If the user–id is missing, the run is automatically rejected.

NOTE:

In this and all other user–id–on configurations, the user–id is required on every batch run. However, if TSS is configured, i.e. LOGONS = 1, the user–id need not be supplied on the @RUN statement for demand runs. In this case, the user–id supplied at log–on will be used.

- Configuration 3

  Quota level one, non–restrictive, user–ids on. All runs are required to specify an account and user–id. The account and user–id of all runs are checked to determine if they are defined in the summary account file. If the account or user–id is undefined, the operator is solicited to determine if the run should be either rejected or the account or user–id entered into the summary account file. Runs with the account or user–id missing are automatically rejected. The maximum priority, assumed priority, real–time level, and deadline values (word five) of the basic accounting block are used at this level.

- Configuration 4

  Quota level one, non–restrictive, user–ids off. The account for each run is checked to determine if it exists in the summary account file. If the account is undefined, the operator is solicited to determine if the run should either be rejected or the account entered into

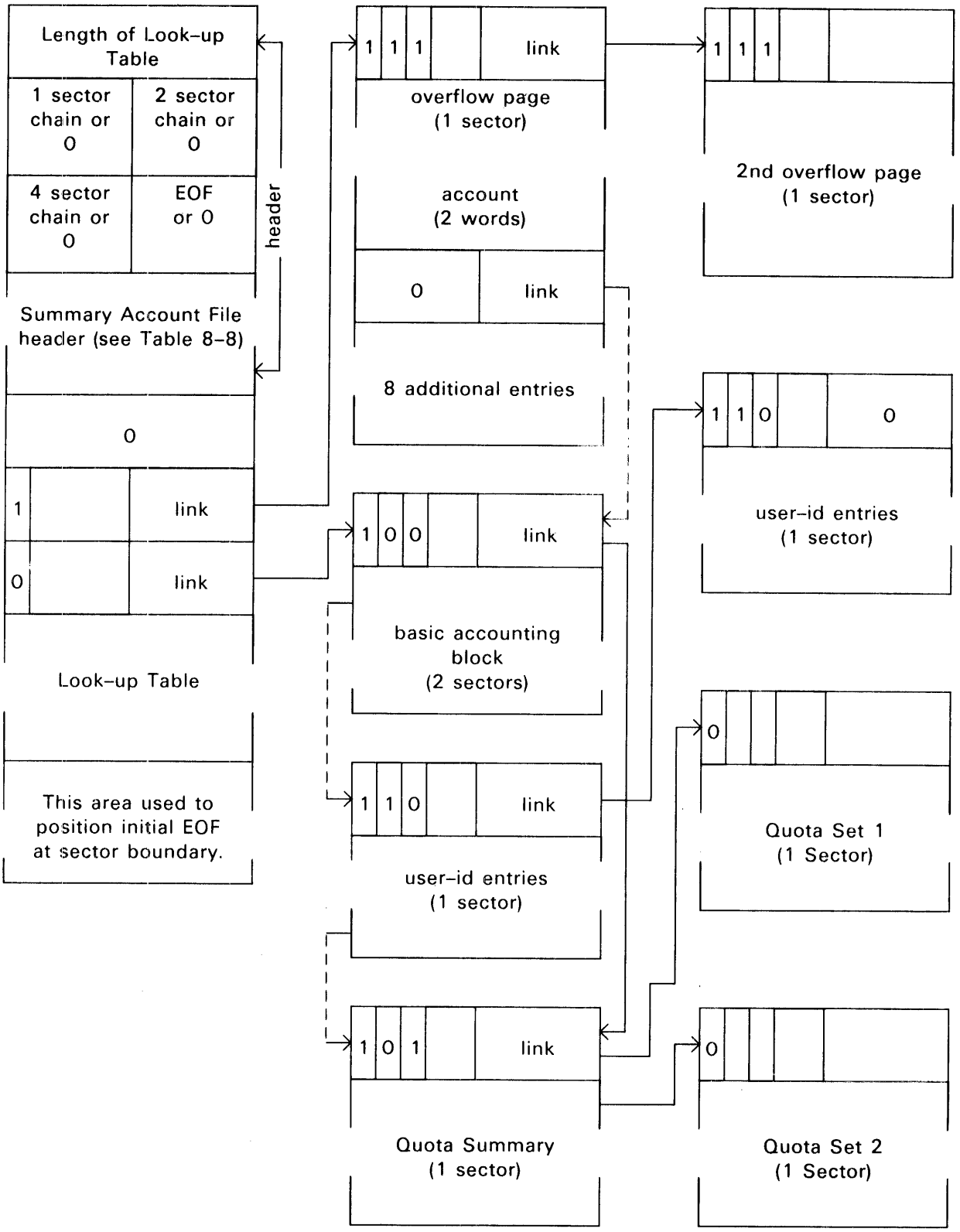


Figure 8-2. Structure of Summary Account File

the summary account file. Runs with no account specified are given the system standard account. If the user-id is specified on the RUN image, it is ignored. Values in word five of the basic accounting block are used at thithe summ

- Configuration 5

  Quota level one, restrictive, user-ids on. All runs must specify an account and user-id. The account and user-id of all runs are checked to determine if they exist in the summary account file. Runs with undefined accounts or user-ids and runs with no account or user-id specified, are automatically rejected. Values in word five of the basic accounting block are used at this level.

- Configuration 6

  Quota level one, restrictive, user-ids off. The account for each run is checked to determine if it is defined in the summary account file. If the account is undefined or if no account is specified, the run is automatically removed. If a user-id is specified on the RUN image, it is ignored. Values in word five of the basic accounting block are used at this level.

- Configuration 7

  Quota level two, non-restrictive, user-ids on. This is configuration 3 plus all of the accumulators in the basic accounting block and user-id entry are maintained.

- Configuration 8

  Quota level two, non-restrictive, user-ids off. This is configuration 4 plus all of the accumulators in the basic accounting block are maintained.

- Configuration 9

  Quota level two, restrictive, user-ids on. This is configuration 5 plus all of the accumulators in the basic accounting block and the user-id entry are maintained

- Configuration 10

  Quota level two, restrictive, user-ids off. This is configuration 6 plus all of the accumulators in the basic accounting block are maintained.

- Configuration 11

  Quota level three, non-restrictive, user-ids on. This is configuration 7 plus all limits in the basic accounting block and user-id entry are enforced.

- Configuration 12

  Quota level three, non-restrictive, user-ids off. This is configuration 8 plus all limits in the basic accounting block are enforced.

- Configuration 13

  Quota level three, restrictive, user-ids on. This is configuration 9 plus all limits in the basic accounting block and user-id entry are enforced.

- Configuration 14

  Quota level three, restrictive, user–ids off. This is configuration 10 plus all limits in the basic accounting block are enforced.

- Configuration 15

  Quota level four, non–restrictive, user–ids on. This is configuration 11 plus all the limits in the Quota set applicable to the run enforced.

- Configuration 16

  Quota level four, non–restrictive, user–ids off. This is configuration 12 plus all limits in the Quota set applicable to the run are enforced.

- Configuration 17

  Quota level four, restrictive, user–ids on. This is configuration 13 plus all limits in the Quota set applicable to the run are enforced.

- Configuration 18

  Quota level four, restrictive, user–ids off. This is configuration 14 plus all limits in the Quota set applicable to the run are enforced.

8.3.5.2. Group Definitions

Under the Quota System, the installation manager can specify what types of equipment are to be grouped together for purposes of control and usage accumulation. This grouping is done at system generation time by the SGS QGROUP. Instructions on the use of this proc are contained in SPERRY UNIVAC 1100 Series Executive, System Generation User Guide, UP–8448 (current version). These group definitions are the ones used for accumulation in the basic accounting block and in the Quota sets. Note that group definitions are required at all Quota levels.

8.3.5.3. Headers

The length of the sections in the basic accounting block and the Quota sets are defined by the headers in these items. The values for the headers are specified by SGS at system generation. After the system is booted, the lengths of the sections cannot be changed. If the recovery boot is done, the header values of the system used to recover must be the same as those in use on the summary account file. If the headers differ, a message is displayed informing the operator that the headers are not the same and then the operator is given the option of either re–initializing the summary account file or doing another recovery boot with a system that has the correct headers.

8.3.5.4. EXEC Defined Accounting Structures (Initialization)

At initial boot time, the EXEC inserts three accounts into the summary account file. The accounts, with their use in the system, are defined as follows:

- System Standard Account

  This account is used to maintain system standard values for the basic accounting block, user–id entry and Quota sets. If an account or user–id is inserted, system standard values

are used for any parameters that are not specified for the basic accounting block or user-id entry. If an account is inserted without specifying a Quota set, the account shares the system standard Quota set. In addition, in a non-restrictive system where user-ids are not configured, all runs that do not specify an account are given the system standard account. Any time that the system standard account is used by a run when user-ids are not configured and the system is non-restrictive, the operator is solicited to determine if the run should be either allowed to continue or be rejected. If the system is restrictive, the run is rejected.

The name of the system standard account and user-id, plus initial values for the basic accounting block, user-id entry and one Quota set are specified by SGS at system generation. After the system is booted, all items of this account can be updated, but the account can never be deleted.

- Overhead Account

  This account is used in ROLOUT, ROLBAK, removed runs and any other EXEC started runs. It can never be used as a user account. User runs attempting to use this account are removed. The name of this account and its user-id are specified by SGS at system generation. This account is initially inserted into the summary account file with the same basic accounting block and user-id values as the system standard account and it shares the system standard Quota set. After the system is booted, all items of this account can be updated, but the account can never be deleted.

- Master Account

  The holder of this account, running with the master user-id, is allowed to perform all of the functions of ER ACCNT$. The name of the master account and user-id are obtained at initial boot time by soliciting the operator for them. If the user-ids are not configured, the operator is solicited for the account only. The master account is initially inserted into the summary account file with the same basic accounting block and user-id values as the system standard account and it shares the system standard Quota set. After the system is booted, all items of this account can be updated, but the account and user-id can not be deleted as long as they are the master account and master user-id.

Note that the EXEC is booted using the overhead account and user-id and that the overhead account has the same initial values as the systems standard account. Therefore, the initial value for the system standard account must be large enough to allow the SYS run to complete. After the SYS run has completed, the overhead account can be unlinked from the system standard account (using QUIP) and the system standard values lowered. The overhead account values must always remain large enough to allow a recovery boot.

8.3.6. Initialization and Recovery

When the EXEC is initially booted, the summary account file, which is named ACCOUNT$R1, is catalogued and assigned exclusively to the EXEC. The header is created, the lookup table is initialized and the accounts described in 8.3.5.4 are inserted into the summary account file.

If a recovery boot is done, the summary account file is assigned exclusively to the EXEC. If it cannot be assigned, the summary account file is re-initialized as described above. If the file can be assigned, a check is made to insure that the basic accounting block headers and the Quota set headers of the system used to reboot match those in use in the summary account file. If they match, all accounts are recovered. If the headers do not match, either the file must be re-initialized as described above, or another reboot must be done, using a system that has correct headers.

If a recovery boot is done with a system having user-ids configured and the system being recovered did not have user-ids configured, the operator is solicited to obtain a master user-id for the current master account. Crossbooting between all other Quota configurations not specifically mentioned above is allowed.

8.3.7. Monitoring and Enforcing Code

8.3.7.1. Run Scheduling

This subsection describes run scheduling as it relates to Quota level.

- Quota Level Zero

  All scheduling information (priority, max pages, etc.) is taken from the SGS if it is not specified on the RUN image. Batch runs are then put in backlog and demand runs are allowed to come active.

- Quota Level One

  The information in word five of the basic accounting block of the account for the run is enforced. The max pages, cards and time values are taken from the configured values if they are not specified on the RUN image. Batch runs are then put in backlog and demand runs are allowed to come active.

- Quota Level Two

  Batch scheduling at this level is the same as at Quota level one. See Quota level two under 8.3.7.2 for additional handling of a demand run.

- Quota Level Three

  Everything done in Quota level one is done here plus the SUA limit is checked to determine if funds are still available for this run. If the user-ids are not configured, only the SUA limit in the basic accounting block is checked. If user-ids are configured and there are SUAs remaining for the account, the user-id SUA limit is checked to determine if funds are available for the user-id. If no funds remain for the account, but the 'can exceed quota' bit is set, scheduling continues as described above. Batch runs for which funds are available are placed in backlog. Demand runs with funds remaining are handled as described under Quota level three in 8.3.7.2.

  If no SUAs remain for the account or user-id, a message is printed informing the user of the lack of funds. Then, if the run is batch, it is removed. If it is demand, the RUN image is rejected, but the terminal remains active.

- Quota Level Four

  Everything done in Quota level three is done here. Additionally, if no Quota set exists for the batch run, the run is removed with a message informing the user that no Quota set exists for the run and the maximum priority value in the set is extracted and used in scheduling. If the Quota set applies to the current time, the run is placed in backlog to await selection. If the Quota set does not apply to the current time, the run is given a start time corresponding to the time of the Quota set and placed in backlog.

If the run is demand and no Quota set exists, the user is informed that there are no Quota sets for demand runs, the RUN image is rejected, and the terminal remains active. If a Quota set exists, but not for the current time, the user is informed that no Quota set exists for the present time and is informed of the time when the next Quota set is available. The RUN image is rejected and the terminal remains active. If a Quota set exists for the present time, the run is handled as described under Quota level four in 8.3.7.2.

8.3.7.2. Run Selection

This section describes run selection as it relates to Quota level. It pertains to batch run after they are selected and to demand runs after the requirements specified for demand runs in 8.3.7.1 are met.

- Quota Levels Zero and One

  No additional Quota–related action is taken at these Quota levels.

- Quota Level Two

  A 31 word PCT buffer is acquired (see 8.3.7.3 for the format). The words pertaining to account fixed mass storage usage are initialized with the values that are currently in the basic accounting block.

- Quota Level Three

  This is level two plus, for batch runs, SUAs remaining are again checked as described under Quota level three in 8.3.7.1. If the account and user–id have SUAs remaining, for batch and demand runs, the number of SUAs remaining are put in word 29 of the 31–word PCT buffer and the run is allowed to come active.

- Quota Level Four

  This is level three plus, for batch runs, a check is made to determine if a Quota set exists for the current time. If no Quota set exists for the current time, the run is removed. If a Quota set exists, maximum time, maximum pages, maximum cards and real–time level values in the Quota set are extracted and used for batch and demand runs. If the Quota set limit on pages, cards, or time is less than the value specified on the RUN image or if the value is not specified on the RUN image, the Quota set value is used. In all other cases, the value specified on the RUN image is used. The run is then allowed to come active.

8.3.7.3. Format of the Quota Information in the PCT

For Quota levels two, three, and four, a 31–word PCT buffer is acquired. The first 28 words contain the applicable Quota set and word 29 contains SUAs remaining. (Words 30 and and 31 are unused). The lower half of PCT word 0213 (EQUFed to RLQ) contains the pointer to the 31–word buffer. The pointer is relative to the start of the PCT.

For Quota level four, the words that pertain to the maximum number of catalogued tracks that a run is allowed to use contain either the value specified in the Quota set, or, if the values remaining in the basic accounting block are less than those in the Quota set, the values in the basic accounting block are used.

8.3.7.4. Enforcement on Facility Assignment

Two areas of facility assignment are enforced.

1. Under Quota level four, a user is allowed to catalogue files with the G or V option if bit 31 of descriptor word one is set. This does not apply to other Quota levels.

2. Under Quota level four, a user is allowed to do an absolute device name assignment or an arbitrary device assignment, or an assignment of a communications device if bit 30 of descriptor word one is set. This does not apply to other Quota levels.

8.3.7.5. Checkpoint/Restart Usage

Under Quota level four, a user is allowed to use Checkpoint/Restart only if bit 29 of descriptor word one is set.

8.3.7.6. ER RSI$

Under Quota level four, a user is allowed to invoke ER RSI$ if bit 28 of descriptor word one is set.

8.3.7.7. SIP Data Reduction

Under Quota level four, a user is allowed to invoke the SIP data reduction programs if bit 27 of descriptor word one is set.

8.3.7.8. SIP ON/OFF

Under Quota level four, a user is allowed to turn SIP on or off via ER SYSBAL$ if bit 26 of descriptor word one is set.

8.3.7.9. Labeled Packs

Under Quota level four, a user is allowed to access labeled packs via ADH if bit 25 of descriptor word one is set.

8.3.7.10. SIP Functions

Under Quota level four, a user is allowed to execute all SIP functions via ER SYBAL$ if bit 24 of descriptor word one is set.

8.3.7.11. Account Sheet

In addition to the information displayed normally, the SUAs used by the run and the SUAs remaining for the account (or user-id if user-ids are configured) are displayed if Quota level one or higher is configured.

8.3.7.12. Summary of Implemented Quota Enforcement Features

Subsections 8.3.7.1 through 8.3.7.11 describe in detail which features of the Quota system are enforceable based on the Quota level configured. This subsection provides a quick reference summary of those features of Quota which are implemented disregarding the effect of Quota level configuration. They are as follows:

- Maximum priority
- Default priority
- Deadline batch allowed
- Real time level
- Whether the account can exceed its SUA Quota
- Maximum SUAs
- User-id submaster controls
- Start and stop times
- Maximum pages output
- Maximum cards output
- Maximum SUPs
- Descriptor bits which allow:

  a. Quota application to online batch
  b. Quota application to remote batch
  c. Quota application to demand
  d. Deadline time
  e. V or G assign option
  f. Absolute device name, arbitrary device or communications device assignment
  g. Checkpoint/Restart
  h. ER RSI$
  i. SIP data reduction
  j. SIP to be turned on or off
  k. Access to labeled packs
  l. All SIP functions

8.3.8. Miscellaneous

When user-ids are configured, every RUN control statement must have a user-id. The user-id field is a subfield of the account field.

This is true for all batch runs. However, when TSS is configured, i.e. LOGONS = 1, the user-id is not required on the demand @RUN statement. In this case, the user-id supplied at log-on is used.

Example:

```
@RUN    RUN1,ACCT/USERID,PROJ
```

When doing a control card or CSF start of a run on a system with user-ids configured, the user-id of the run doing the start is used as the user-id of the run being started. For the start ST keyin, the user-id must be specified as a subfield of account field.

When doing a control card or CSF restart of a run on a system with user-ids configured, the user-id of the run doing the restart is used as the user-id of the restarted run. For the restart RS keyin, the user-id must be specified as a subfield of the account field.

8.4. TAPE LABELING SYSTEM

8.4.1. Introduction

This subsection defines the tape labeling system. This system incorporates the American National Standard Magnetic Tape Labels for Information Interchange X3.27-1969 to provide a comprehensive tape labeling facility.

The intent of a tape labeling facility is twofold. First, an installation may desire tape labeling to minimize the impact of operator and user error. This error can be either intentional or unintentional. Secondly, a site may desire to provide a facility which will allow a smooth data transfer between computing systems via magnetic tape. These computing systems may be logically close (may be under common managerial control) or they may be quite apart. In either case, a need exists to provide a functional interface between data on one system and data on a different system. In this instance, data interchange is the primary objective of a tape labeling facility.

One of the primary objectives of tape labeling is to provide a way that will allow a specific site to define its position with respect to the extremes of data integrity/security and data interchange. The first step in providing this facility is defined in this section as a 'common ground'. This 'common ground' is referred to as the base level throughout this section.

The base level of the tape labeling facility in the Operating System delivered to a site creates, maintains and validates certain label records within the constructs of the American National Standard. Other optional label records can be processed depending on site implementation.

8.4.1.1. Definitions

This section provides definitions of the terms used in connection with tape labeling and the labels used by both the system and the user.

a

One of the following characters: the digits 0,1,...,9, the upper case letters A,B,...,Z, and the following special characters:

(Space) ! " % & ' ( ) * + , - . / : ; < = > ?

The ASCII characters were chosen from the center four columns of the code table specified in ASCII, except for those positions where there is a provision for alternative graphic representation. (See Table 8-15.)

Block

A group of contiguous characters recorded on and read from magnetic tape as a unit. A block may contain one or more complete records.

Double Tape Mark

A delimiter, consisting of two tape marks (see definition below), that is used to indicate the end of a volume or of a file set. It also occurs when an empty file section or an empty file exists on a volume, in which case they are not interpreted as a double tape mark but rather as two single tape marks framing an empty file section. "Empty" here means that no blocks are present between the tape mark following the beginning–of–file section label group and the tape mark preceding the end–of–file section label group or end–of–file label group of that file section or file.

File

A collection of information in data blocks, consisting of records pertaining to a specified subject and delimited by tape marks and label groups. The absence of information or data blocks results in the creation of a null or empty file, that is, a file without information.

File Section

That part of a file recorded on any one volume. The sections of a file may not have sections of other files interspersed.

File Set

A collection of one or more related files recorded on one or more volumes. A file set may consist of:

- One file recorded on a single volume
- More than one file recorded on a single volume.
- One file recorded on more than one volume.
- More than one file recorded on more than one volume

Header Label Group

A label group consisting of a system volume header label (if it is the first label group on the volume), system file header labels, and a tape mark. Optionally and under the control of the user, the group may also contain user file header labels following the appropriate system labels.

Label

An 80 character block at the beginning or end of a volume or file serving to identify and/or delimit that volume or file. It is not considered part of the file section it delimits.

Label Group

A collection of contiguous labels pertaining to a file which precedes or follows that file or part of that file on a volume. The volume header label(s) and the following file header label(s) are considered to be the first label group on the volume.

Label Identifier

A name consisting of three alphabetic characters identifying the type of label. It is followed by a label number.

Label Number

A numeric character in the range 1 to 9 following the label identifier.

n

Any numeric digit in the range 0 through 9.

System Labels

Those labels controlled by the Operating System and which the user may not access directly. System end-of-volume and system end-of-file labels are mutually exclusive in any given end label group. The system labels are:

- VOL1 Volume header label
- HDRn File header label. Must be in consecutive ascending order and symmetric to EOF/EOV labels
- EOFn End-of-file label (corresponds to HDRn with the same rules applying)
- EOVn End-of-volume label (corresponds to HDRn with the same rules applying)

Tape Mark

A special code configuration or sequence recorded on magnetic tape indicating the boundary between files and labels and also between certain label groups. The presence of the tape mark is generally detected by the tape unit or by the tape unit controller and is thereby signalled to the Executive. (See the section on double tape marks).

User Labels

Those labels whose reading, writing, and contents (except for the initial label identifier and number) are controlled by the user. They are:

- UHLa User file header label. No order or symmetry of occurrence is necessary
- UTLa User Trailer Label. Also known as user end-of-volume or user end-of-file labels, depending on the circumstances. (No correspondence to UHLa is necessary.)

Volume

A physical unit of storage media. In this specification the medium is magnetic tape, so a volume is a reel of magnetic tape.

8.4.1.2. Structure of Magnetic Tape File

SINGLE FILE, SINGLE VOLUME

VOL1 HDR1 HDR2*—file A—*EOF1 EOF2**

SINGLE FILE, MULTIVOLUME

VOL1 HDR1 HDR2*—first part file A—*EOV1 EOV2**

VOL1 HDR1 HDR2*—last part file A—*EOF1 EOF2**

MULTIFILE, SINGLE VOLUME

VOL1 HDR1 HDR2*—file A—*EOF1 EOF2* HDR1 HDR2*—file B *EOF1 EOF2**

MULTIFILE, MULTIVOLUME

VOL1 HDR1 HDR2*—file A—*EOF1 EOF2*HDR1 HDR2*—first part file B—*EOV1 EOV2**

VOL1 HDR1 HDR2*—continuation of file B—*EOV1 EOV2**

VOL1 HDR1 HDR2*—last part of file B—*EOF1 EOF2*HDR1 HDR2*—file C—*EOF1 EOF2**

NOTE:

Asterisk (*) indicates tape mark and two asterisks (**) indicate two tape marks.

8.4.1.3. Format of Labels

8.4.1.3.1. Volume Header Label (VOL 1)

Character Position	Field Nbr	Contents	Length (Char)
1–3	1	VOL	3
4	2	1	1
5–10	3	Reel Number	6
11	4	Accessibility Code	1
12–31	5	Spaces	20
32–37	6	Spaces	6
38–51	7	Owner Identification	14

Character Position	Field Nbr	Contents	Length (Char)
52–79	8	Spaces	28
80	9	Label Standard Version	1

Note:

All characters are ASCII.

8.4.1.3.2. First File Header (HDR1)

Character Position	Field Nbr	Contents	Length (Char)
1–3	1	HDR	3
4	2	1	1
5–21	3	File Identifier	17
22–27	4	Set Identification	6
28–31	5	File Section Number	4
32–35	6	File Sequence Number	4
36–39	7	Generation Number	4
40–41	8	Generation Version Number	2
42–47	9	Creation Date	6
48–53	10	Expiration Date	6
54	11	Accessibility	1
55–60	12	Block Count	6
61–73	13	System Code	13
74–80	14	Reserved for Future Standardization	7

8.4.1.3.3. Second Header (HDR2)

Character Position	Field Nbr	Contents	Length (Char)
1–3	1	HDR	3
4	2	2	1
5	3	Record Format	1
6–10	4	Block Length	5
11–15	5	Record Length	5
16–50	6	Reserved for Operating Systems	35
51–52	7	Buffer Offset	2
53–80	8	Reserved for Future Standardization	28

8.4.1.3.4. First End of File Label (End of Volume)

Character Position	Field Nbr	Contents	Length (Char)
1–3	1	EOF (EOV)	3
4	2	1	1
5–54	3–11	Same as Corresponding Fields in First File Header Label	50
55–60	12	Block Count	6
61–80	13–14	Same as Corresponding Fields in First File Header Label	20

8.4.1.3.5. Second End–of–File Label (Second End–of–Volume)

A second end–of–file label (end–of–volume) contains EOF (EOV2) as the label identifier and label number and contains the same information as fields three through eight of the HDR2 label.

8.4.1.3.6. Optional Labels

Character Position	Field Nbr	Contents	Length (Char)
1–3	1	UHL or UTL	3
4	2	Label Number	1
5–80	3	User Option	76

8.4.1.4. Label Field Definitions

8.4.1.4.1. Volume Header Label Set

The volume header label set is composed of one label record, the VOL1 record. This record is created, maintained, and verified by the base level (see 8.3.1) and is always the first record of a volume. The fields of the VOL1 record and their significance to the base level are:

Character Position: 1 to 3

Field Name: Label Identifier

Length: 3 Characters

Description:

The contents of this field are always VOL. This string is the first indication to the Executive Operating System that the volume is labeled. The contents of this field are verified, created, and maintained by the base level.

Character Position: 4

Field Name: Label Number

Length: 1 digit

Description:

The contents of this field are always 1. This string is the second indication to the Executive Operating System that the volume is labelled. Label records with a label identifier of VOL and a label number other than 1 are prohibited by the base level.

Character Position: 5 to 10

Field Name: Volume Identifier

Length: 6 'a' characters

Description:

The contents of this field are assigned by the owner of this volume to distinguish it from other volumes under the owner's control. The base level maintains and verifies this field. The contents are created by the base level using information provided by the site. The base level requires this field to contain the reel number that corresponds with that used on the user's @ASG statement.

Character Position: 11

Field Name: Accessibility

Length: 1 'a' character

Description:

The contents of this field indicate restrictions, or the lack thereof, on access to the information in this volume. The base level defines two values for the contents of this field. One value (a space) indicates unlimited accessibility. The other value (a comma ',') restricts access of this volume to users with the same account number as that contained in the owner identification field of this label record. All other values are treated as a space by the base level. Values reserved for site purposes are: A B C D E F G H I J K L ! " $ % & '

Character Position: 12 to 37

Field Name: (reserved for future standardization)

Length: 26 spaces

Description:

The base level does not place any characters other than spaces in this field. If upon verification the Operating System notes non-spaces in this field, it ignores them.

Character Position: 38 to 51

Field Name: Owner Identification

Length: 14 'a' characters

Description:

The contents of this field identify the owner of the volume. The base level normally expects spaces in this field. If the accessibility field of this record contains a comma, this field is expected to contain the account number of the owner of this volume. If the accessibility field does not contain a comma, the contents of this field are ignored by the base level.

Character Position: 52 to 79

Field Name: (reserved for future standardization)

Length: 28 spaces

Description:

The base level inserts spaces in this field. On verification, the base level ignores this field.

Character Position: 80

Field Name: Label Standard Version

Length: 1 digit

Description:

The contents of this field indicate the Standard to which this volume conforms. A '1' indicates the American National Standard.

8.4.1.4.2. System File Header Label Set

The system file header label set is created, verified, and maintained by the base level. The base level maintains two records of this label set, the HDR1 and HDR2. Other HDR labels are recognized but ignored by the base level.

8.4.1.4.2.1. The HDR1 Label Record

Character Position: 1 to 3

Field Name: Label Identifier

Length: 3 characters

Description:

The string HDR must occur in this field for the base level to recognize this record as part of the system file header label set.

Character Position: 4

Field Name: Label Number

Length: 1 digit

Description:

The base level recognizes any valid label number, but it only processes and creates records whose label number is either '1' or '2'. All others are ignored.

Character Position: 5 to 21

Field Name: File Identifier

Length: 17 'a' characters

Description:

This field's contents identify the file to its owner. The base level creates, verifies, and maintains the contents of this field. The base level, in the normal case, inserts and verifies a string which identifies the qualifier and filename used on the user's @ASG statement that created this file. If the number of character positions of the qualifier and filename (the sum of these two) exceed 16 positions, the right most portion of the qualifier is truncated, but the entire filename is always maintained. For example, if the actual qualifier and filename used was: MYPROJECTID*CREATEDFILE, the base level inserts MYPRO*CREATEDFILE into this field. In the default case, the same character string occurs in this field for all files within a file set. In other words, all HDR1 records for all files on a volume or multiple volumes contain the same file identifier in the default case. The file identifier in the default case is left justified, space filled.

Character Position: 22 to 27

Field Name: File Set Identifier

Length: 6 'a' characters

Description:

This field identifies this file set among other file sets. The base level inserts, on output, the information from the volume identifier field of the VOL1 record of the first or only volume of the file set into this field. The contents of this field are not verified by the base level.

Character Position: 28 to 31

Field Name: File Section Number

Length: 4 'n' digits

Description:

This field identifies this section among the sections of this file. The base level inserts the correct number into this field in the case of multiple volume files. No verification is performed upon this field.

Character Position: 32 to 35

Field Name: File Sequence Number

Length: 4 'n' digits

Description:

This field identifies the placement of this file among the other files of this file set. The file sequence number of the first file of a set is 1 and it is incremented by 1 for each additional file. The base level creates and maintains the contents of this field.

Character Position: 36 to 39

Field Name: Generation Number

Length: 4 'n' digits

Description:

This field distinguishes among successive generations of this file. The base level inserts, on output, the absolute F-cycle of the file into this field. The base level does not verify the contents of this field.

Character Position: 40 to 41

Field Name: Generation Version Number

Length: 2 'n' digits

Description:

The contents of this field are to be used to distinguish successive iterations of the same generation. As this field has no significance to 1100 Series Operating Systems, it is ignored by the base level on input. On output, the value of zero is inserted.

Character Position: 42 to 47

Field Name: Creation Date

Length: 6 characters

Description:

The contents of this field indicate the date upon which this file was created. The field's construction is a space followed by two 'n' digits for the year and three 'n' digits indicating the day within the year. This field is maintained and created by the base level.

Character Position: 48 to 53

Field Name: Expiration Date

Length: 6 characters

Description:

The contents of this field indicate the date upon which this file is considered expired by its owner. The field's construction is the same as for the creation date field. The base level creates (using user provided information), maintains, and verifies this field.

Character Position: 54

Field Name: Accessibility

Length: 1 'a' character

Description:

The contents of this field indicate restrictions, or the lack thereof, on access to the information in this file. The base level provides a limited number of predefined values for this field. This field is created, maintained and verified by the base level. The base level defines three definitions for this field. These definitions are not mutually exclusive. Any, none, or all of these definitions may be used by a user. A 2^0 indicates that the verification of the file identifier field of this record is required. A 2^2 indicates this file to be a READ ONLY file. A 2^1 indicates the file to be a WRITE ONLY file. This value, or combination of values, is added to 060 to form accessibility characters 061–067. Any value not recognized by the base level will result in the following: (1) File identifier verification not required, (2) file is not READ ONLY, (3) file is not WRITE ONLY. Values reserved for site use are: A B C D E F G H I J K L ! " $ % & '

The above values of 061–067 will be determined by the R, W, and F options on the assign card; i.e., if the 'R' option is on the assignment when the file is created, the file is read only until it expires. Table 8–14 shows HDR1 Accessibility codes.

Table 8–14. HDR1 Accessibility Codes

Accessibility Character	Assign Options		
	F	R	W
040	X		
061			
062	X	X	
063		X	
064	X		X
065			X
066	X	X	X
067		X	X

Character Position: 55 to 60

Field Name: Block Count

Length: 6 zero digits

Description:

The purpose of this field is to maintain a correspondence with the same field in the EOF1 and/or EOV1 record. The base level creates this field with zeros and maintains this value.

Character Position: 61 to 73

Field Name: System Code

Length: 13 'a' characters

Description:

The contents of this field indicate the system that recorded this file. The left most 7 positions of this field are created, maintained, and verified by the base level. These 7 positions are used to identify label records written by the base level and the Univac update version to the base level. The remaining 6 positions are reserved for use by the site to identify its tape labeling facility implementation. In the base level the left most characters are 'U1100–1'.

Character Position: 74 to 80

Field Name: (reserved for future standardization)

Length: 7 spaces

Description:

The base level inserts spaces in this field. On verification, the base level ignores this field.

8.4.1.4.2.2. The HDR2 Label Record

Character Position: 1 to 3

Field Name: Label Identifier

Length: 3 characters

Description:

The string HDR must occur in this field for the base level to recognize this record as part of the system file header label set.

Character Position: 4

Field Name: Label Number

Length: 1 digit

Description:

To be recognized as a HDR2 record by the base level, this field must contain a '2'.

Character Position: 5

Field Name: Record Format

Length: 1 'a' character either F, D, S, or U

F = Fixed Length
D = Variable with number of characters in the record specified in decimal.
V = Variable with the number of characters specified in binary.
U = Unspecified

Description:

This field defines the record format of the data file. As applied to 1100 Series Systems, this field has little significance. On output, in the default case, the base level enters a 'U' in this field. On input, the base level ignores the contents of this field. A user execution level facility is provided in the base level to allow a user to provide one acceptable value for this field. On input, the facility allows a user to retrieve the contents of this field.

Character Position: 6 to 10

Field Name: Block Length

Length: 5 'n' digits

Description:

The field specifies the maximum number of characters per block of the data file. As applied to 1100 Series Systems, this field has little significance. On output, in the default case, the base level inserts the maximum possible value in this field (99999). On input, this field is ignored. A user execution level facility is provided in the base level to allow a user to provide a valid value for this field. This facility also allows a user to retrieve the contents of this field.

Character Position: 11 to 15

Field Name: Record Length

Length: 5 'n' digits

Description:

This field defines the logical record length of records in the data file. As applied to 1100 Series Systems, this field has little meaning. On output, in the default case, the base level inserts zeros into this field. A user execution level facility is provided in the base level to allow a user to provide a valid value for this field. This facility also allows a user to retrieve the contents of this field.

Character Position: 16 to 50

Field Name: (reserved for system use, of which positions 16–32 are reserved for site use.)

Character Position: 51 to 52

Field Name: Buffer Offset Length

Length: 2 'n' digits

Description:

This field specifies the length in characters of any additional field inserted before the first record in a data block. As applied to 1100 Series Systems, the field has little significance. On output, in the default case, the base level inserts zeros in this field. On input, this field is ignored by the base level. A user execution level facility is provided by the base level to allow a user to provide a valid value for this field. This facility also allows a user to retrieve the contents of this field.

Character Position: 53 to 80

Field Name: (reserved for future standardization)

Length: 28 spaces

Description:

The base level inserts spaces in this field on output. On verification, the base level ignores this field.

8.4.1.4.3. Header Labels 3-9

The base level does not provide for the implementation of HDR 3-9 labels. The base level bypasses them during label checking if they exist.

8.4.1.4.4. User File Header Label Set

The records that compose the user file header label set are those label records whose label identifier is the string UHL. They are the responsibility of the user. The base level provides an execution level facility for a user to create and process records from this label set. On output, the base level insures that two fields (the label identifier and label number field) contain valid values. The correct placement of these records after the system file header label set is also insured by the base level. On input, these records are ignored by the base level.

Character Position: 1 to 3

Field Name: Label Identifier

Length: 3 characters

Description:

The base level verifies the user has provided the string UHL in this field during creation.

Character Position: 4

Field Name: Label Number

Length: 1 digit

Description:

The only allowable values for this field are 1 through 9. The base level verifies the validity of this field. The order of the records using this field is not verified.

Character Position: 5 to 80

Field Name: (reserved for user application)

Length: 76 'a' characters

Description:

The user may provide any 'a' character information desired in this field.

8.4.1.4.5. System End-of-File Label Set

The system end-of-file label set is composed of label records whose label identifier field contains the string EOF. The base level supports two records from this set: the EOF1 and EOF2 records. The EOF1 is an exact duplicate of the HDR1 record of the file with the exception of the label identifier and block count fields. The block count field of the HDR1 record is always '00000'. The block count field of the EOF1 record has exactly the same character position and length as the field in the HDR1, but it contains the number of data blocks since the beginning of the data file. The base level validates this field on input and logs any error to the user.

A user execution level facility is provided to allow a user to provide and retrieve the contents of this field.

The EOF2 record is an exact duplicate of the HDR2 record of the file with the exception of the label identifier field.

The EOF label set is used primarily during READ BACKWARD operations. The base level performs file verification using the HDR label set if the user is attempting WRITE or READ FORWARD operations. The EOF label set is used during file verification if the user is attempting READ BACKWARD operations.

8.4.1.4.6. System End-of-Volume Label Set

The system end-of-volume label set is composed of label records whose label identifier field contains 'EOV'. The base level supports two records from this label set: EOV1 and EOV2. The discussion of the system end-of-file label set also applies to this label set. The format and description are identical to those of the end-of-file label records.

8.4.1.4.7. User Trailer Label Set

The records that compose the user trailer label set are those label records whose label identifier consists of the string UTL. They are the responsibility of the user. The discussion of user file header label records above is also applicable to the discussion of user trailer label records.

8.4.2. Reading and Writing Tape Label Blocks (TLBL$)

The ER TLBL$ enables the user to read or write selected portions of HDR1, HDR2, EOF1, EOF2, EOV1, and EOV2 and allows the user to read and write user labels before and after each file. See SPERRY UNIVAC 1100 Series Executive System Volume 2, EXEC Programmers Reference UP-4144.2 (current version) for details of reading and writing of tape label blocks.

8.4.3. Multivolume Processing

On output if an ER TSWAP$ is requested the tape file is closed with

```
*EOV1  EOV2**
```

If the last I/O function to the tape was a write tape mark, a void file is created before closing the tape.

```
*HDR1  HDR2**EOV1  EOV2**
```

The information used to create the label is the last information received by the system to write or verify labels. Therefore, if new information is required in the end of volume labels or if user trailer labels are desired, the ER TLBL$ requests must be made prior to the ER TSWAP$ request.

8.4.4. Reverse Processing

The reverse processing of label groups is handled the same as forward processing. The direction of processing only determines which labels to check after a tape mark is encountered or which labels to check before the user's next I/O.

8.4.5. Interchangeable Tapes

Only 9-track tapes are considered interchangeable tapes. All labels are written at the density the user assigns (800 fpi, 1600 fpi or 6250 fpi), without the use of translators. If format A is assigned (quarter word mode) on a UNISERVO 12, 14, 16, 20, 30, 32, 34, 36 tape unit, the labels are written as 80-character blocks. If format A is not assigned, labels are written as 81-character blocks (an 80-character label and 1-character padding).

On input, a labeled tape file written at a non-Sperry Univac site is accepted to be read at anytime but only when it expires is it allowed to be overwritten. All security checks (file identification) are bypassed.

8.4.6. Seven Track Tape Drives

All labels are 8–bit ASCII characters written in format C (8–bit packed) at the density chosen by the installation; either medium or high density with odd parity.

All tapes assigning low density require unlabeled tapes and automatically receive the 'J' option on the assign.

8.4.7. Unlabeled Tapes

Unlabeled tapes are allowed in the tape labeling system by setting the SGS Parameter TLSIMP to 1 or the 'J' option on the assignment.

8.4.8. End of Tape Substatus

An end–of–file status will also receive a substatus when the end of tape is detected. This status is a 3 in the AFC field (S3 of word 3) of the I/O packet.

8.4.9. Forward and Backward Space Functions on VIC and VIIIC

Use of the forward space file function (FSF$) and backspace file (BSF$) is expanded to UNISERVO VIC and VIIIC tape drives. This is necessary to bypass label validation until the tape is moved to the file which is to be accessed; i.e., a tape which has six files, all with different filenames, can have the sixth file accessed without requiring validation information from the user for the first five files.

When a FSF$ or BSF$ function is the user's first access to a data file, tape labeling bypasses label checking and allows the tape to be moved to the next end–of–file. As soon as a function is issued other than FSF$ or BSF$, tape labeling can label check the header labels on a forward function or trailer labels on a backward function and allow or prohibit access to the data.

8.4.10. Error Conditions and Messages

Console error messages occur only when label checking the first label group on a tape. The messages occur if the wrong reel is mounted, if a blank was requested and the reel mounted was not a logical blank, if the assignment allows only labeled tapes and an unlabeled tape is mounted, if an unlabeled tape is required and a labeled tape is mounted, or if the first HDR1 label cannot be found.

8.4.10.1. File Access Inhibited Errors

Tape files are set READ and WRITE inhibited if the following errors occur.

1. 'E' answer to tape labeling error messages.
2. Fail file identifier validation.
3. Any I/O error processing a label group beyond the first label group where a label cannot be recognized.
4. An unrecoverable I/O error such as down status or run termination.

Any access to a READ and WRITE inhibited file receives an I/O error status 020 (contingency type 1) which is recoverable if a contingency routine is registered.

If READ and WRITE inhibit is set on the first label group, the following error message is logged:

devnam ACCESS NOT ALLOWED ON *aaa*

Where *devnam* is the name of the tape drive of the assign and *aaa* is the reel number.

All tape labeling log messages are type 13 and appear with the accounting information of the run that assigned the tape.

If READ and WRITE inhibit is set beyond the first label group, the following message is logged:

*qualifier*filename (fff)* ACCESS NOT ALLOWED ON FILE *n*

Where *qualifier*filename (fff)* is the external filename and f-cycle of the tape file. '*n*' is the file on that tape file in which the error occurred.

8.4.10.2. Other Log Messages

Errors which occur beyond the first label group are logged to the user. The following messages are used.

*qualifier*filename (fff)* { EOF1 / EOF2 / HDR1 / HDR2 } LABEL NOT FOUND FOR FILE *n*

*qualifier*filename (fff)* FILE *n* EXPED *x* BLOCKS REC *y*

*qualifier*filename (fff)* I/O ERROR *z* ON { R / W } ON { TRL / HDR } LABEL ON FILE *n*

where '*x*' is the number of blocks in the EOF1, EOV1, or HDR1 labels (HDR1 in reverse processing), '*y*' is the number of blocks actually read when the tape mark was received. *z* is the actual I/O error status received (other than 0), 'R' and 'W' indicate READ or WRITE, 'TRL' indicates trailer label group, and 'HDR' indicates header label group. 'EXPED' refers to expected and 'REC' refers to received. accounting information of the run that assigned the tape.

The first two error log messages are informative error messages indicating something is missing on the tape. In both cases recovery is attempted. In the case where a label is missing the system tries to determine if any types of labels exist. If so, it tries to position accordingly. If not, it will reposition after the last valid label group or at the tape mark read by the user. In the case where file *n* expected *x* blocks and received *y*, the system has picked the block count in the EOF1 or EOV1 label, compared it to the number of blocks extended and found them not equal.

In the case where the number of blocks read by the user does not correspond with the block count in the EOF1, EOF2, or HDR2 the error message is logged to the user and processing continues.

If an I/O error is detected on a read of a label beyond the first label group, the input buffer is checked to see if there was a partial read of a label. If there was, processing is attempted. If there was not, the file is set READ and WRITE inhibited and the user is informed that access is not allowed on that file.

8.4.11. American Standard Code for Information Interchange

The 7-bit code in Table 8-15 may be extended to 8-bits by the addition of columns 8 (bit pattern 1000_2) through 15 (bit pattern 1111_2) to the right of the table as shown and the further addition of bit position 2^7 with a binary value of 0 to columns 0 through 7 shown below.

This coded character set is to be used for the general interchange of information among information processing systems, communications systems, and associated equipment.

Table 8-15. American Standard Code for Information Interchange (ASCII)

		Control Characters		Graphic Characters					
	Col.	0	1	2	3	4	5	6	7
Row	Bits	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	END	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
10	1010	LF	SUB	*	:	J	Z	j	z
11	1011	VT	ESC	+	;	K	[	k	{
12	1100	FF	FS	,	<	L	\	l	|
13	1101	CR	GS	-	=	M	]	m	}
14	1110	SO	RS	.	>	N	^	n	~
15	1111	SI	US	/	?	O	_	o	DEL

9. Logging and Accounting

9.1. LOGGING

9.1.1. Introduction

Extensive logging and accounting capabilities are incorporated into the Executive system to collect information pertaining to each run and to certain general Executive actions, such as I/O errors. The information can be used for accounting and general postprocessing purposes. A summary accounting file containing information about each account is maintained and updated automatically by the system at the termination of each run. In addition, a master log of all logging and accounting information is produced.

A utility processor, LOGFED (see 9.2), is provided to edit the master log file. The LOGFED processor serves only to list the contents of one or more master log files and is provided only as an example to the installation manager to aid in writing routines meaningful to the installation.

Figure 9–1 is a block diagram of the logging and accounting process.

9.1.2. Log Entry Initiation and Control

Log entries are initiated by the Executive at various significant points throughout a run in order to obtain and preserve in the master log a chronological record of the activities of the system. In addition, the user run can enter messages into the master log. Console messages are also entered. The total information gathered is of value not only for accounting (billing) purposes, but also to capture the unique actions of a particular run and to evaluate system performance.

The information generated falls into the following categories:

- Run initiation
- User-specified log control statement (@LOG) from either a run stream or submitted via a CSF$ request.
- Program initiation and termination
- I/O errors

- Console activity
- Symbiont activity
- Tape labeling information
- Checkpoint/Restart information
- Facility usage
- Run termination
- Catalogued mass storage file usage
- Software detected errors
- Hardware fault information
- Common bank reload activity

The log control routine controls the flow of all logging information that is generated. As each logging request is made, the log control routine chains the new log entry information by run in the temporary log file or chains the log information to the EXEC chain and records it directly in the master log file.

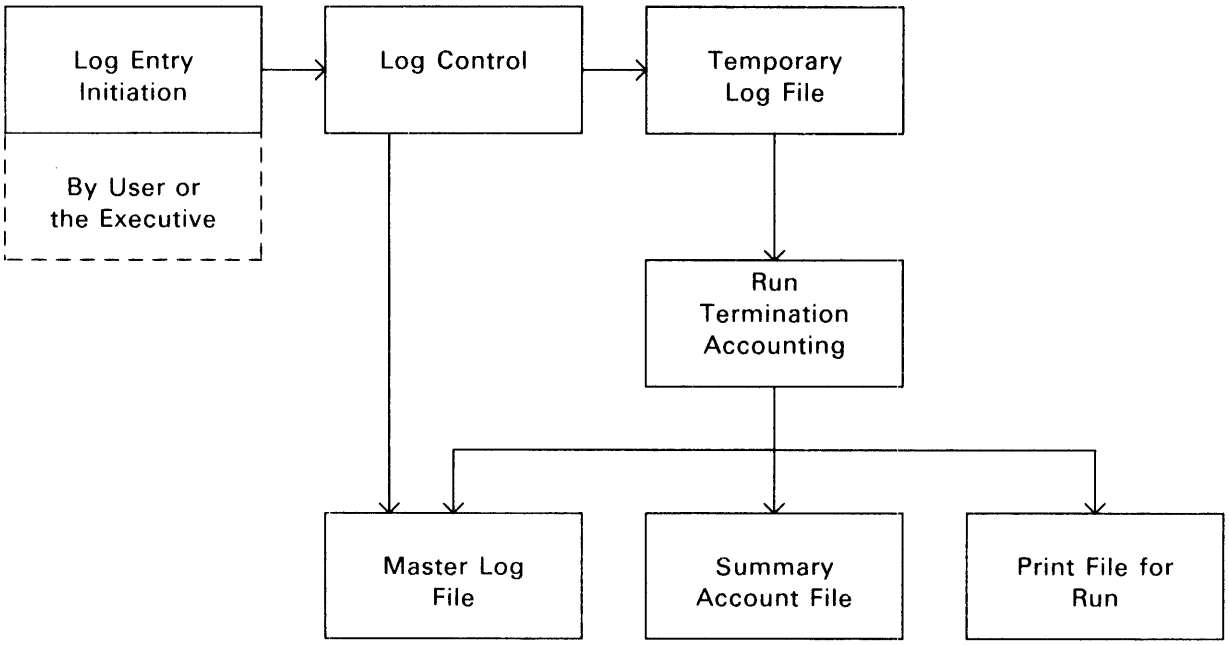


Figure 9-1. Logging and Accounting Process

9.1.3. Print File Output

At each run termination, information which pertains to the run is placed at the end of the PRINT$ file by the run termination accounting routine. The information presented is as follows:

- Run identity
- Control language log statements
- Console messages pertaining to the run
- Executive Request log statements
- Project identity
- Account number
- Total run time (SUPs)
- CPU SUPs
- Control card and ER SUPs
- Core block SUPs
- I/O SUPs
- Wait time
- Standard Units of Accounting (SUAs) used by the run
- SUAs remaining for the account or user-id
- Pages of print produced by the run
- Number of card images read by the run and punched by the run
- Time and date of run initiation
- Time and date of run termination

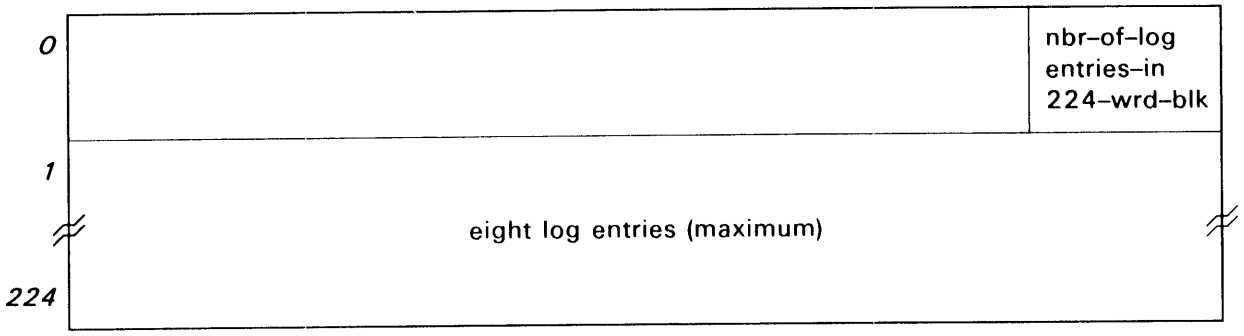
9.1.4. Summary Account File Creation and Updating

A second function of the run termination accounting routine is to update the totals in the summary account file for the given account. This is done in the course of creating the master log file, since each log entry is moved from the temporary log file to the master log file.

The summary account file is a FASTRAND-formatted mass storage file which is set up at system initialization and permanently assigned to the Executive system. An entry exists in the summary account file for each allowable account number. Totals are kept for each account until cleared by a file processing routine or until the file is replaced during initial system loading.

9.1.5. Master Log File Creation and Control

The run termination accounting routine inserts entries into the master log file (SYS$*SYSTEM$LOG$). This file is catalogued either on tape or on FASTRAND-formatted mass storage (specified at system generation). The file is made up of a number of 224-word blocks each containing up to a maximum of eight log entries. The number of these blocks depends upon the length of the file set at system generation. The blocks have the following format:



This file is initially catalogued by the Executive during the initial boot from tape. Log entries are written on the file until the preset maximum length of the file is encountered. Then the current F-cycle of the file is released and a new one is started. This switch enables the user to assign and read the file just released. An operator keyin (LS) also causes the Executive to switch files. In case of I/O errors on either tape or mass storage, the Executive performs an automatic switch to obtain a new file. When the F-cycle limit is reached, the system drops the oldest cycle to make room for the newest F-cycle. If the file is on tape and an end-of-tape is encountered, TSWAP$ is called for a new reel of tape and an F-cycle change does not occur.

At system initialization, the file is catalogued with a (-0) relative F-cycle. If the file (-0) is on FASTRAND-formatted mass storage, the user should use caution in assigning and reading this file since the Executive will be writing on it at the same time. However, if the file is on tape, the user is not able to assign and read this F-cycle until the Executive releases it. When a switch occurs, the Executive frees the current (-0) F-cycle and catalogues a new (-0) F-cycle. The former (-0) F-cycle becomes relative (-1) and is available to the user. If the file is on mass storage, the (EOF) sentinel is one word of all 7's (octal) at the beginning of the block. If the file is on tape, a hardware EOF is written before doing either a TSWAP or the switch.

The particular log entries found in the master log are described in detail in 9.1.6. The LOGFED routine is described in 9.2.

9.1.6. File Formats

Each value in a master log file entry is binary unless it is obviously a symbolic name such as program name, version name, run-id, qualifier, filename, project-id, or account number, in which case it is in Fieldata that is left-justifed and space filled. Several items in all entries have a common format and are described here to facilitate easier understanding of the following entry formats. The common items are as follows:

- date and time
  - This is the format of time and date as received from a call to TDATE$.

- message
  - Refers to a string of Fieldata characters, the maximum length of which is specified in the entry format.
- system indicator
  - An indicator which is used by log editing programs to distinguish changes in log file formats produced by different levels of the Executive.
- nbr-of-log-entries
  - Used only for the first entry in a 224-word block.
- nbr-of-words-in-entry
  - A count of words used in the entry exclusive of words 0, 25, 26 and 27.

The different types of log entries which may appear in the master log are described in 9.1.6.1 through 9.1.6.30.

Each of the entry formats contains a word that provides either the date and time of the log entry or the date and time of some program action. The format of this word is:

mm/dd/yy/seconds-after-midnight

where the year (yy) is modulo 64.

The log file contains a log file header which is created on the initial access to the log file. The purpose of this header is to present a relationship between types of peripheral equipment, i.e. specific equipment mnemonics, and the equipment codes found in the log entry.

Figure 9-2 illustrates the format of the log file header. Note that it is organized into two sections; a Table of Contents (TOC) section and a table section. The descriptive information in the header is contained within the tables where each table contains a discrete set of information. The TOC section and each table will be 28 words or a multiple of 28 words in length. The unused portions of the TOC and each table are zero filled.

The TOC section contains the following information:

1. The identifier word 'SYSLOG',
2. Sector address of first log entry,
3. Number of tables, and
4. Two word descriptors for each table.

Each descriptor contains a Fieldata table name, the starting address of the table and the length of the used portion of the table. The starting address is a word address which is relative to the start of the header. The table length is specified in number of words. The TOC is currently limited to 224 words or 110 tables.

At present, only the equipment mnemonic table is defined. The Fieldata table name for this table is 'EQPMNM'.

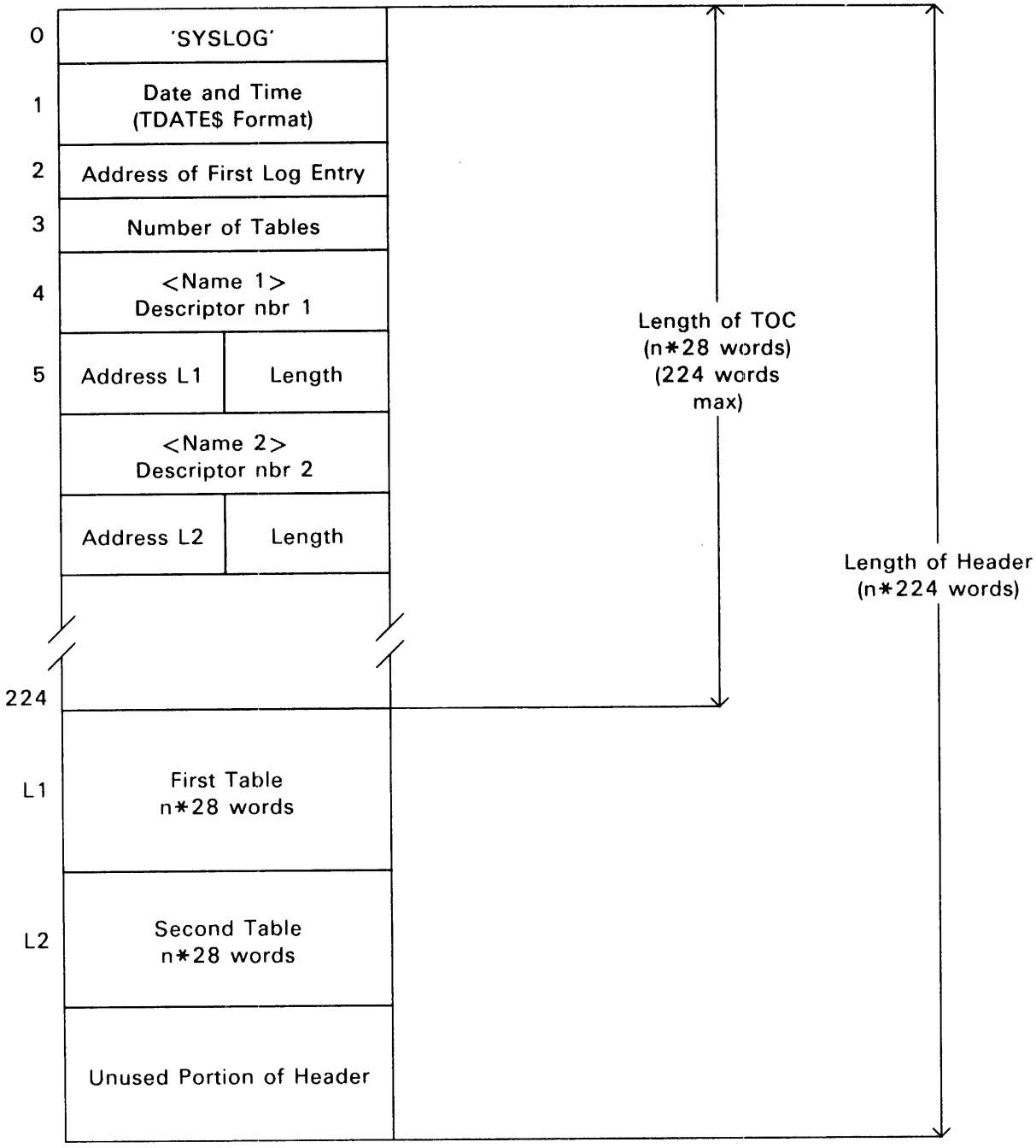


Figure 9-2. Log File Header Format

9.1.6.1. Control Statement Log Entries

Log entries specified by the @LOG control statement are placed in the master log file in the order in which they occur. The entry format is shown in Table 9-1.

Table 9-1. Control Statement Log Entries

0	*entry-type (1)*	*nbr-of-wrds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*message (22-word maximum)*					
24						
25	*date-and-time-of-log-entry*					
26	*reserved*					
27	*run-id*					

9.1.6.2. Facility Usage Log Entries

Whenever the configuration of a run is changed by assigning a tape or arbitrary device file, an entry is made in the master log file. The entry format is shown in Table 9-2.

Table 9-2. Facility Usage Log Entries

0	*entry-type (2)*	*nbr-of-wrds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*equipment code*		*reserved*	*TODRUM-addr-for-assign/free of-a-communications-device*		
2	*logical-device-name*					
3	*entry-2*					
4						
23						

Table 9-2. Facility Usage Log Entries (continued)

24	*SUPs + voluntary-delay-time*
25	*date-and-time-of-log-entry*
26	*reserved*
27	*run-id*

NOTES:

1. For a communications device assign the logical device name replaced by the TODRUM address.

2. Words 1-2 are for entry 1, words 3-4 are for entry 2, ..., words 21-22 are for entry 11.

9.1.6.3. Catalogued Mass Storage File Usage Entry

Whenever a catalogued file is created, assigned, or freed (using a @FREE control statement or at run termination) a log entry is created and subsequently inserted into the master log. The entry format is shown in Table 9-3 and described below.

Table 9-3. Catalogued Mass Storage File Usage Entry

0	*entry-type (3)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1 2	*qualifier*					
3 4	*filename*					
5 6	*project-id*					
7 8	*account-number*					
9	*unused*	*flag 2*	*current-assigns*	*absolute-F-cycle*		
10	*date/time-of-FREE-or-0*					

Table 9-3. Catalogued Mass Storage File Usage Entry (continued)

11	*date/time-of-cataloguing*	
12	*date/time-of-ASG,A-or-0*	
13	*Words 13 through 20 contain count of file granules at time of log entry creation which exist on mass storage devices having equipment codes 030 through 037.*	
20		
21	*equipment-code*	*unused*
22	*unused*	
23	*unused*	
24	*SUPs + voluntary-delay-time*	
25	*date-and-time-of-log-entry*	
26	*reserved*	
27	*run-id*	

Word 9

current assigns — The number of current assignments for this file when the log entry is created.

flag 2

040 Position granularity
020 Private file
010 File is being dropped
004 File was or is being assigned with exclusive use
002 Write only file
001 Read only file

9.1.6.4. Program Termination Log Entry

For each program in the run, termination information is entered in the master log. The entry format is shown in Table 9-4.

Table 9-4. Program Termination Log Entry

0	*entry-type (4)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*program-name*					
2						
3	*version-name*					
4						
5	*program-termination-date/time*					
6	*CPU-time (1106/1108, 1100/10, 1100/20) in-integer-binary or primary-storage-SRC-in-floating-point (1110 1100/40)*					
7	*negative-zero (1108) or extended-storage-SRC-in-floating-point (1110, 1100/40) or quantum-timer-value-in-100 nanoseconds-intervals (1100/80)**					
8	ER-and-control-card-charge (200 *μsec intervals)**					
9	*Words 9 through 18 contain I/O transfer counts for Group 1 through Group 10 I/O devices**					
18						
19	SUPs (200 *μsec intervals)**					
20	*CBSUPs (core-block-SUPs)**					
21	voluntary-delay-time (200 *μsec intervals)**					
22	time-in-real-time-mode (200 *μsec intervals)**					
23	*unused*			*last-reentry-address*		
24	*condition-word*					
25	*date-and-time-of-log-entry*					
26	*reserved*					
27	*run-id*					

* The values for CPU time, SRC counts, ER and control card charges, I/O transfer counts, voluntary delay time, SUPs, and CBSUPs are run.

* The values for CPU time, SRC counts, ER and control card charges, in the Type 4 and the Type 16 log entries reflect what has accumulated in the run up to the time the log entry

is built. In computing the quantity used in the execution of a program, it is necessary to find the differences between the corresponding values in the Type 4 and Type 16 log entries for the program.

9.1.6.5. Run Termination Log Entry

At the completion of each run, termination information is entered in the master log. The entry format is shown in Table 9-5 and described below.

Table 9-5. Run Termination Log Entry

0	*entry-type (5)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*account-number*					
2						
3	*project-id*					
4						
5	*run-initiation-time (TDATE$ format)*					
6	*run-termination-time (TDATE$ format)*					
7	*cards-in*			*cards-out*		
8	*priority*	*page-count-using-standard-page-size*				
9	*estimate-run-time-in-SUPs*					
10	*actual-run-time-in-SUPs*					
11	*core-block-SUPs*					
12	*Words 12 through 19 contain the parameters Tracks*(SUPs + voluntary-delay-time) for each of the fixed mass storage device groups. Tracks applies to temporary files and expansion of catalogued files assigned to the run.*					
19						
20	*user-id*					
21						
22	*total-number-of-sectors allocated/released*			*total-number-of allocation/release-calls*		
23	*granule-table-r/w*			*total-number-of directory-control-calls*		

Table 9-5. Run Termination Log Entry (continued)

24	*total-number-of-PRINT$-pages*	*PRINT$-pages-since-last-BRKPT*
25	*date-and-time-of-log-entry*	
26	*reserved*	
27	*run-id*	

Word 23

granule-table-r/w — A count of the read/write operations used to maintain granule tables.

total-directory control-calls — The total count of references to the Executive cataloguing routines to support all catalogued files in the run.

9.1.6.6. I/O Error Log Entry

A record of all I/O errors is kept in the master log. A count of valid references to a unit is maintained in main storage and, when an error occurs, this information along with the error information is placed in the master log. The reference count is cleared to zero at that time. Note that after a predetermined number of retries (number of retries is device dependent), all of which fail, the operator is notified and operator intervention is required. The entry formats are shown in Tables 9-6, 9-7, and 9-8 and described below.

Table 9-6. Channel Program Error Unsolicited Interrupt Log Entry (1100/80)

0	*entry-type (6)*	*nbr-wrds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*error-type (0 or 1)*	*sense byte-count*	*unit-equipment-type*		*control-unit equipment-type*	
2	*unit-device-name*					
3	*control-unit-device-name*					
4	*user function code*	*handler function code*	*nbr-of-lost-log-entries*		*retry-count (neg-if-retry-failed)*	
5	*I/O-path-used*			*system type*	*error-message index*	
6	*number-of-references-since-last-try*					

Table 9-6. Channel Program Error Unsolicited Interrupt Log Entry (1100/80) (continued)

7	channel-command-word-0
8	channel-command-word-1
9	channel-status-word-0
10	channel-status-word-1
11	channel-status-word-2
12	up-to-six-words-of-sense-bytes
17	
18	up-to-six-words-of-device-dependent-information
23	
24	reel-id-if-tape, pack-id-if-disk
25	date-and-time-of-log-entry
26	absolute address of first command CCW in this segment
27	run-id-name

Table 9-7. Channel Program Error Unsolicited Interrupt (non-1100/80)

0	entry-type (6)	nbr-wrds in-entry	system indicator	reserved	reserved	nbr-of-log entries-in 224-wrd-blk
1	error-type	sense byte-count	unit-equipment-type		control-unit equipment-type	
2	device-name					
3	control-unit-name					
4	user function code	handler function code	nbr-of-lost log-entries		retry-count (neg-if-retry-failed)	

Table 9-7. Channel Program Error Unsolicited Interrupt (non-1100/80) (continued)

5	*I/O-path-used*	*system type*	*error-message-index*
6	*number-of-references-since-last-try*		
7	*access-control-word*		
8	*chain-pointer-word*		
9	*EI-status-word*		
10	*MSA-auxiliary-status*		
11	*logical control unit number* / *logical-MSA number*	*unused*	
12 ≈ 17	*up-to-six-words-of-sense-bytes*		
18 ≈ 23	*up-to-six-words-of-device-dependent-information*		
24	*reel-id-if-tape, pack-id-if-disk*		
25	*date-and-time-of-log-entry*		
26	*absolute address of first command ACW in this segment*		
27	*run-id-name*		

Word 1

error-type	0	channel program error
	1	unsolicited interrupt
	2	nonsense interrupt

Word 4

system-type	0	EON
	1	ON40
	2	ON80

Words 18–23

If the associated device is a tape, then the data in words 18 through 23 has the following format:

18	*nbr-of-files-extended*	*channel program condition code*	*nbr-of-blocks-extended*
19	*reserved*		
20 21	*command-bytes (2-words)* *or 2 EF words*		
22	*reserved*		
23	*reserved*		

If the associated device type is mass storage, then the data in words 18 through 23 has the following format:

18	*mass-storage error-code*	*retry-count-on error-type*	*word/record from-USTWDR*
19	*device relative word address*		
20 21	*command-bytes (2-words)* *or 2 EF words*		
22	*error offset from label*	*label (continued to next word)*	
23	*reserved*		

If the associated device is a symbiont type, then the data in words 18 through 23 has the following format:

18	*run-id (print only)*	
19		*symbiont equipment index*
20	*command-bytes (2-words)*	
21	*or 2 EF words*	
22	*reserved*	
23	*reserved*	

Table 9-8. Nonsense Interrupt Log Buffer Format

0	*entry-type (6)*	*nbr-of-wrds in-the log-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*error-type 2*	*nbr-of interrupts in-the log-entry*	*nbr-of interrupts thrown away*		*nbr-of-interrupts-in-stack*	
2	*reserved*					
3	*nonsense-interrupt-save-area(20-words)*					
≈	*(up-to-five-4-word-entries)*					
22						
23	*reserved*					
24	*reserved*					
25	*date-and-time-of-log-entry*					
26	*reserved*					
27	*run-id*					

9.1.6.7. Console Log Entries

Each console message is placed in the master log. At run termination, every message pertaining to the run is printed at the end of the program listing. The entry format is shown in Table 9–9.

Table 9–9. Console Log Entries

0	*entry–type (7)*	*nbr–of–wds in–entry*	*system indicator*	*reserved*	*reserved*	*nbr–of–log entries–in 224–wrd–blk*
1					*msg–nbr*	
2	*message (23–word–maximum)*					
24						
25	*date–and–time–of–log–entry*					
26	*reserved*					
27	*run–id*					

9.1.6.8. Checkpoint Log Entry

When a checkpoint is used in a run, an entry is made in the master log with pertinent information concerning the checkpointed run. The entry format is shown in Table 9–10.

Table 9–10. Checkpoint Log Entry

0	*entry–type (8)*	*nbr–of–wds in–entry*	*system indicator*	*unused*	*unused*	*nbr–of–log entries–in 224–wrd–blk*
1	*message (24–word maximum)*					
24						

Table 9-10. Checkpoint Log Entry (continued)

25	*date-and-time-of-log-entry*
26	*reserved*
27	*run-id*

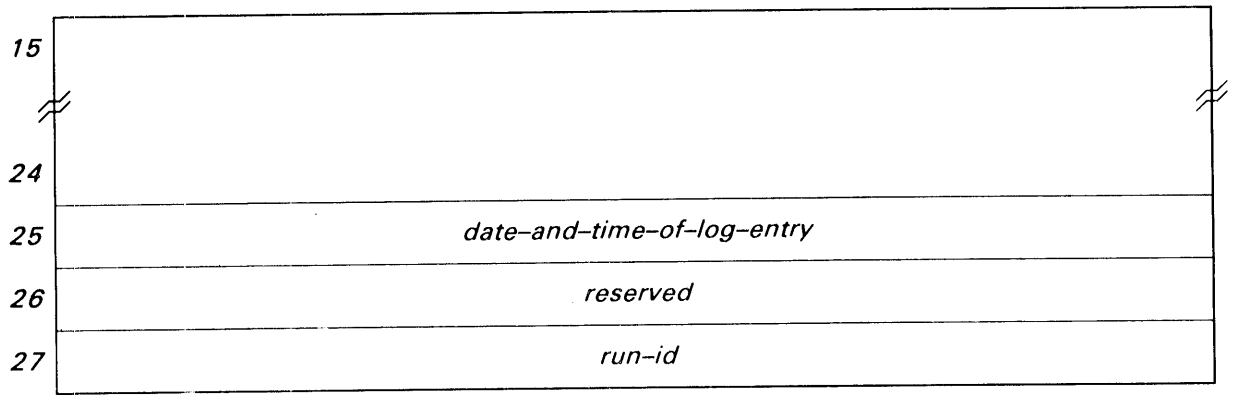
9.1.6.9. Run Initiation Log Entry

When a run is opened, an entry is made in the master log with the pertinent information concerning the run. The entry format is shown in Table 9-11 and described below.

Table 9-11. Run Initiation Log Entry

0	*entry-type (9)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*A*	*priority*	*start-time (in minutes)*		*deadline-time (in minutes)*	
2	*estimated-pages-out*			*estimated-card-out*		
3	*run-id (new or generated)*					
4	*run-id (old or original)*					
5	*project-id*					
6						
7	*account-number*					
8						
9	*sequence-id*					
10	*run type*			*estimated-run-time-(SUPs)*		
11	*device-association*					
12	*user-id*					
13						
14	*SUAs remaining at run initiation*					

Table 9-11. Run Initiation Log Entry (continued)



Word 1

A — The possible values are:

010 – T option is specified on @RUN control statement
004 – P option is specified on @RUN control statement
002 – C option is specified on @RUN control statement
001 – S option is specified on @RUN control statement

Word 9

sequence-id — If the run is from a batch input device, this parameter is the run-id of the preceding run from the same device. If the run was scheduled via @START, this parameter is 0.

Word 10

run type — The possible types are

4 – demand
5 – deadline batch
6 – batch

Word 11

device-association — This parameter is the Fieldata name of the device which read the run, or, in the case of a run scheduled via @START, it is the Fieldata name of an onsite input device.

This entry is always the first one in the first block of a series of contiguous blocks in the master log file for a given run. The run-id field (new or generated) contains the same identity as word 27 of each entry for the subject run.

9.1.6.10. Console Replies Log Entry

Replies to console type and read messages are placed in the master log. The replies as well as the type and read messages are printed at the end of the program listing. The entry format is shown in Table 9-12.

Table 9-12. Console Replies Log Entry

0	*entry-type (10)*	*nbr-of-words in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1				*msg-nbr*		
2	*message (11-word-maximum)*					
24						
25	*date-and-time-of-log-entry*					
26	*reserved*					
27	*run-id*					

9.1.6.11. Log Keyin Entry

Each time a LG keyin is input from the operators console, the message will be recorded in a log entry. The entry format is shown in Table 9-13 and described below.

Table 9-13. Log Keyin Entry

0	*entry-type (11)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*message (9-word-maximum)*					
24						

Table 9-13. Log Keyin Entry (continued)

25	*date-and-time-of-log-entry*
26	*reserved*
27	*EXEC 8*

NOTE:

Word 27 will be "EXEC 8" if no run-id is specified on the LG keyin.

9.1.6.12. Unsolicited Keyin Log Entry

The entry format is shown in Table 9-14.

Table 9-14. Unsolicited Keyin Log Entry

0	*entry-type (12)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1				*actual-keyin-in-Fieldata-format (left-justified)*		
2	*message (23-word-maximum)*					
24						
25	*date-and-time-of-log-entry*					
26	*reserved*					
27	*EXEC 8*					

9.1.6.13. Tape Labeling Log Entry

When the tape labeling feature of the Executive is used, log entries are made in the master log. These entries contain pertinent information concerning allocation and release of tape reels, and errors encountered during tape labeling. The entry format is shown in Table 9-15.

Table 9-15. Tape Labeling Log Entry

0	*entry-type (13)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*message (24-word-maximum)*					
24						
25	*date-and-time-of-log-entry*					
26	*reserved*					
27	*run-id*					

9.1.6.14. Symbiont End of Processing Log Entry

When a symbiont finishes processing an input or output file, an entry is made in the EXEC chain on the master log with pertinent information concerning the processed file. The entry format is shown in Table 9-16 and described below.

Table 9-16. Symbiont End of Processing Log Entry

0	*entry-type (14)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*equipment-code*		*file-type*			
2	*symbiont-name*					
3	*line-count-or-cards-in/out*					
4	*run-id-of-associated-run*					
5						
24						
25	*date-and-time-of-log-entry*					

Table 9-16. Symbiont End of Processing Log Entry (continued)

26	*reserved*
27	*EXEC 8*

Word 1

equipment-code — equipment code of symbiont device

file type — has value of:

01 – input cards
02 – output cards
03 – output pages

9.1.6.15. Symbiont Start of Processing Log Entry

When a symbiont begins processing an input or output file, an entry is made in the EXEC chain on the master log with pertinent information regarding the file to be processed. The entry format is shown in Table 9-17 and described below.

Table 9-17. Symbiont Start of Processing Log Entry

0	*entry-type (15)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*equipment-code*		*file-type*			
2	*symbiont-name*					
3						
4	*run-id-of-associated-run*					
5	*account-number*					
6						
7	*project-id (build-if-generated-by-symbiont)*					
8						

Table 9-17. Symbiont Start of Processing Log Entry (continued)

9	
//	//
24	
25	*date-and-time-of-log-entry*
26	*reserved*
27	*run-id*

Word 1

equipment-code — equipment code of symbiont device

file-type — has value of:

01 - input cards
02 - output cards
03 - output pages

9.1.6.16. Program Initiation Log Entry

When a program is initiated, a log entry is made to record pertinent values that are cumulative during the run. The entry format is shown in Table 9-18.

Table 9-18. Program Initiation Log Entry

0	*entry-type (16)*	*nbr-of-words in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*program name*					
2						
3	*version-name*					
4						
5	*program-initiation-date/time*					
6	*CPU-time (1106/1108, 1100/10, 1100/20) in-integer-binary or primary-storage-SRC-in-floating-point (1110, 1100/40)*					

Table 9-18. Program Initiation Log Entry (continued)

7	*negative-zero (1108) or extended-storage-SRC-in-floating-point (1110, 1100/40) or quantum-timer-value-in-100-nanoseconds-intervals (1100/80)*
8	ER + control-statement-charge (200 *μsecs)*
9	*I/O-transfer-count-group-1-devices*
//	//
18	*I/O-transfer-count-group-10-devices*
19	SUPs (200 *μsecs)*
20	*CBSUPs (core-block-SUPs)*
21	voluntary-delay-time (200 μsecs)
22	time-in-real-time-mode (200 μsecs)
23	*unused*
24	*condition-word*
25	*date-and-time-of-log-entry*
26	*reserved*
27	*run-id*

9.1.6.17. Run Termination Supplement Log Entry

The entry format is shown in Table 9-19.

Table 9-19. Run Termination Supplement Log Entry

0	*entry-type (17)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*CPU-time (1106/1108, 1100/10, 1100/20) in-integer-binary or primary-storage-SRC-in-floating-point (1110, 1100/40)*					
2	*negative-zero (1108) or extended-storage-SRC-in-floating-point (1110, 1100/40) or quantum-timer-value-in-100-nanosecond-intervals (1100/80)*					
3	ER-and-control-card-charge (200 *μsecs)*					

Table 9-19. Run Termination Supplement Log Entry (continued)

4	*I/O-transfer-count - group-1-devices*
13	*I/O-transfer-count - group-10-devices*
14	voluntary-delay-time (200 *μsecs)*
15	1110-CAU-time 1100/80 CPU-time (200-*μsec-increments)*
16	*SUAs-used*
17	
24	
25	*date-and-time-of-log-entry*
26	*reserved*
27	*run-id*

9.1.6.18. Recovery Close-Out Log Entry

When a system recovery is performed before a run has gone through run termination accounting, the sequence of log entries for that run may not contain the type 17 log entry which effectively closes out the set of log entries for that run. In this case, the following log entry is added to the set of log entries at the time of system recovery. This log entry effectively closes an open set of log entries. The entry format is shown in Table 9-20.

Table 9-20. Recovery Close-Out Log Entry

0	*entry-type (18)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	"SYSTEM HAS ABORTED. LOG RECOVERED AT: *hhmmss*"					
6						
7						

Table 9-20. Recovery Close-Out Log Entry (continued)

8	
≈	≈
24	
25	*date-and-time-of-log-entry*
26	*reserved*
27	*run-id*

9.1.6.19. Cooperative Accounting

When a PRINT$ or PUNCH$ file is breakpointed (@BRKPT), or when an alternate print or punch file is closed, a log entry will be inserted into the log. The entry format is shown in Table 9-21 and described below.

Table 9-21. Cooperative Accounting

0	*entry-type (19)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*PRLEN*		*PRCUR*		*PRTOP*	*PRBOT*
2	*print/punch-control-calls*			*pages/cards*		
3	*file-type-indicator*			*pages/cards since last BRKPT*		
4	*filename*					
5						
6						
≈	≈					≈
24						
25	*date-and-time-of-log-entry*					
26	*reserved*					
27	*run-id*					

Word 1

PRLEN	Current length of page less bottom margin
PRCUR	Current position on page
PRTOP	Current top margin
PRBOT	Current bottom margin

Word 3

File type indicator	Type 2 for punch file, type 3 for print file.

9.1.6.20. Facility Usage Summary Log Entry

The facility usage summary log entry (Table 9-22) is added to the user chain after the type 17 if any tape, removable disk, or symbiont units were assigned by the run.

Table 9-22. Facility Usage Summary Log Entry

0	*entry-type (20)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*tape-unit-SUPs-in-seconds (Group 1)*					
	≈					≈
	tape-unit-SUPs-in-seconds (Group n)					
	disk-unit-SUPs-in-seconds (Group 1)					
	≈					≈
	disk-unit-SUPs-in-seconds (Group n)					
	symbiont-unit-SUPs-in-seconds (Group 1)					
	≈					≈
23	*symbiont-unit-SUPs-in-seconds (Group n)*					
24	*nbr-of-tape-group-entries*		*nbr-of-disk-group-entries*		*nbr-of-symbiont group-entries*	
25	*date-and-time-of-log-entry*					
26	*reserved*					
27	*run-id*					

9.1.6.21. Symbiont Close-Out Log Entry

When the last output file for a run has been processed, an entry is made in the EXEC log chain. The entry format is shown in Table 9-23.

Table 9-23. Symbiont Close-Out Log Entry

0	*entry-type (21)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*run-id-of-associated-run*					
2						
≈						
24						
25	*date-and-time-of-log-entry*					
26	*reserved*					
27	*EXEC 8*					

9.1.6.22. EXEC Segment Validation Log Entry

When an error is encountered while loading an EXEC segment, a log entry is made and subsequently placed in the Master Log. The entry format is shown in Table 9-24.

Table 9-24. EXEC Segment Validation Log Entry

0	*entry-type (22)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*mass-storage-address-of-segment*					
2	*keyword-or-checksum-received*					
3	*'$FUNC$'-or-checksum-expected*					
4	*ANS = (A,B,or G)*		*number-of-attempts-to-load-segment*			
5	*segment-name*					

Table 9-24. EXEC Segment Validation Log Entry (continued)

6	
≈	≈
24	
25	*date-and-time-of-log-entry*
26	*reserved*
27	*EXEC 8*

A type 22 log entry is also generated when an error is encountered reading a mass storage master bit table. In this case the entry format is shown in Table 9-25.

Table 9-25. Mass Storage Master Bit Table Error

0	*entry-type (22)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*mass-storage-address-of-MBT*					
2	*checksum-expected*					
3	*checksum-received*					
4	*reserved*					
5	*'MBTERR'*					
6						
≈						≈
24						
25	*date-and-time-of-log-entry*					
26	*reserved*					
27	*EXEC 8*					

9.1.6.23. Software Instrumentation Package Log Entry

The type 23 log entry has been reserved for use by Software Instrumentation Package (SIP). Entries are variable in length and always constitute one physical block of the log file on tape (see 5.1).

9.1.6.24. Software Detected Error Log Entry

Type 24 has been reserved for software detected error log entries. (See Tables 9–26 through 9–32.) Several different entries (and formats) have been defined and are described below.

The log entry shown in Table 9–26 is created by FREL whenever a track of mass storage to be released by FREL is not allocated to the file.

Table 9–26. Software Detected Error Log Entry (Subcode 0001)

0	*entry–type (24)*	*nbr–of–wds in–entry*	*system indicator*	*reserved*	*reserved*	*nbr–of–log entries–in 224–wrd–blk*
1					*subcode (0001)*	
2	*'FREL'*					
3	*qualifier*					
4						
5	*filename*					
6						
7	*user–run–id*					
8	*assign–options–from–file–item*					
9	*initial–reserve*			*maximum–granules*		
10	*highest–track–referenced*			*highest–granule–assigned*		
11	*count of granules released*			*rel–cycle number*	*absolute cycle–number*	
12	*pack–id*					
13	*caller*					
14		*MBT flag*	*FRESC call*	*unit–status–table*		
15	*equipment–type*		0=FASTRAND 1 = drum		*removable if nonzero*	*00 = TRK 40 = POS*

Table 9-26. Software Detected Error Log Entry (Subcode 0001) (continued)

16	logical device name		
17	IDL-location		
18	number-of-tracks-released		nbr-of-positions now-available
19	current track-nbr		
20	starting-track-nbr-being-released		
21	original-request-limits		
22	bit-pattern-for-position-where		
23	release-cannot-be-made		
24	request-limits-at-time-of-malfunction		
25	date-and-time-of-log-entry		
26	reserved		
27	EXEC 8		

Table 9-27. Software Detected Error Log Entry (Subcode 0003)

0	entry-type (24)	nbr-of-wds in-entry	system indicator	reserved	reserved	nbr-of-log entries-in 224-wrd-blk
1					subcode (0003)	
2	'UPROAR' or 'UPREG'					
3	pack-id					
4	qualifier					
5						
6	filename					
7						

Table 9-27. Software Detected Error Log Entry (Subcode 0003) (continued)

8	*project-id-on-the-pack*		
9			
10	*account-on-the-pack*		
11			
12	*time-of-last-reference*		
13	*time-of-cataloging*		
14	*f-cycle*	*0*	*nbr-times-assigned*
15	*equipment-type*		
16	*project-in-the-MFD*		
17			
18	*account-in-the-MFD-or-0*		
19			
20	*time-of-last-reference-or-0*		
21	*time-of-cataloging-or-0*		
22	*equipment-type*		*nbr-of-times-assigned-or-0*
23			*error-address*
24			
25	*date-and-time-of-log-entry*		
26	*reserved*		
27	*EXEC 8*		

When a removable disk pack is being registered and a directory track is lost, the following log entry is created. The log entry format is shown in Table 9-28.

Table 9-28. Software Detected Error Log Entry (Subcode 0004)

0	*entry-type (24)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1					*subcode (0004)*	
2	*'UPROAR'*					
3	*removable-disk-pack-id*					
4	*initial-directory-track-address*					
5	*status*	*I/O function*				
6	*access-word*					
7	*disk/drum-address*					
8				*error-address*		
9	*first-word-initial-DAS*					
10	*last-word-initial-DAS*					
11	*current-word-current-DAS*					
12						
≈						
24						
25	*date-and-time-of-log-entry*					
26	*reserved*					
27	*EXEC 8*					

When a removable disk pack cannot be registered due to some unrecoverable error, the following log entry is created. The log format is shown in Table 9-29.

Table 9-29. Software Detected Error Log Entry (Subcode 0005)

0	*entry-type (24)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1					*subcode (0005)*	
2	*'UPROAR'*					
3	*pack-id*					
4	*initial-directory-address*					
5	*status*	*I/O function*				
6	*access-word*					
7	*disk/drum-address*					
8				*error-address*		
9	*directory-lock-word (FIFPTS)*					
10	*FIPTLK*			*FATULK*		
11						
≈						
24						
25	*date-and-time-of-log-entry*					
26	*reserved*					
27	*EXEC 8*					

The log entry shown in Table 9-30 is created when a C/SP illegal function or communications error is encountered.

The purpose of this log entry is to record information about fatal system errors, non-fatal system errors, and system boots.

Table 9-30. Fatal, Non-Fatal System Errors and System Boots Log Entry

0	*entry-type (24)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1					*subcode (0006)*	
2	*XERCODE*			*XERSCODE*		
3	*XERCPU*	*XERTYP*		*XERPADD*		
4	*XERDATE*			*XERTIME*		
5	*XERSES*			*XERSEQ*		
6	*XERJKS*			*XERSEG*		
7		*XERBTP*	*XERBTC*	*XERNSES*		
8						
9	*XERDEP*					
10						
≈						
24						
25	*date-time-of-log-entry*					
26	*reserved*					
27	*EXEC 8*					

where:

The value of the field XERTYP indicates the contents of the log entry:

XERTYP	Log Entry Contents
0	Fatal system error information and boot information
1	Non-fatal system error information
2	Only boot information

The fields of the log entry have the following meaning for each value of XERTYP:

Field	XERTYP 0	1	2	Definition
XERCODE	X	X		System error code
XERSCODE	X	X		System error subcode
XERCPU	X	X		CPU number of reporting CPU
XERPADD	X	X		Address from which error was reported
XERTIME	X			Time in seconds since midnight of fatal error
XERDATE	X			Date of fatal error mm dd yy octal
XERSES	X	X		Session in which error occurred
XERSEQ	X	X		Sequence number of latest non-fatal error
XERSEG	X	X		Segment index of a function that reports an error
XERDEP	X			Contains information that is dependent upon the type of fatal system error
XERBTP	X		X	Boot type 0 = drum boot, 1 = recovery boot, 2 = auto boot, 3 = tape dump 4 = hardware boot
XERJKS	X		X	Jump key settings on reboot, master bit format
XERBTC	X		X	Boot CPU number
XERNSES	X		X	Number of the session initiated by the boot

The communications software error log entry format is shown in Table 9-31.

Table 9-31. Software Detected Error Log Entry (Subcode 0007)

0	*entry-type (24)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wd-blk*
1					*subcode (0007)*	
2	*word-1-of-status-information*					
3	*word-2-of-status-information*					
4	*word-3-of-status-information*					
5						
≈						
24						

Table 9-31. Software Detected Error Log Entry (Subcode 0007) (continued)

25	*date-and-time-of-log-entry*
26	*reserved*
27	*EXEC 8*

The C/SP error log entry has the format shown in Table 9-32 and described below.

Table 9-32. Software Detected Error Log Entry (Subcode 0073)

0	*entry-type (24)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1			*LGTYPE*	*C/SP-nbr*	*subcode (0073)*	
2	*QCCI0*					
3	*QCCI1*					
4	*QCI0 Data ...*					
24						
25	*date-and-time-of-log-entry*					
26	*reserved*					
27	*EXEC 8*					

Word 1

LGTYPE
- 0 C/SP initiated log entry
- 1 Illegal function from the C/SP
- 2 Communications Error

9.1.6.25. Checkpoint Initiation Log Entry

Whenever a CKPT operation is initiated, a log entry is created and subsequently placed in the Master Log. The entry format is shown in Table 9-33.

Table 9-33. Checkpoint Initiation Log Entry

0	*entry-type (25)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*primary-storage-reference-count*					
2	*primary-storage-reference-count*					
3	*extended-storage-reference-count*					
4	*extended-storage-reference-count*					
5	*ER+control-card-charge*					
6	*I/O-SUPs*					
15						
16	*voluntary-delay-time*					
17	*CPU-time*					
18						
24						
25	*date-and-time-of-entry*					
26	*reserved*					
27	*run-id*					

9.1.6.26. Restart Initiation Log Entry

Whenever a restart (RSTRT) operation is initiated a log entry is made and subsequently inserted into the Master Log. The entry format is shown in Table 9-34.

Table 9-34. Restart Initiation Log Entry

0	*entry-type (26)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*primary-storage-reference-count*					
2	*primary-storage-reference-count*					
3	*extended-storage-reference-count*					
4	*extended-storage-reference-count*					
5	*ER+control-card-charge*					
6	*I/O-SUPs*					
15						
16	*voluntary-delay-time*					
17	*CPU-time*					
18						
24						
25	*date-and-time-of-entry*					
26	*reserved*					
27	*run-id*					

9.1.6.27. Hardware Fault Log Entry

Tables 9-35 through 9-39 are the log entries for 1110, 1100/40 and 1100/80 systems hardware interrupts where recovery is attempted:

Whenever one of these interrupts occur, an entry will be made in the master log. However, if the error is fatal, a log entry will not be made.

Table 9-35. I/O Fault Log Entry (Non-1100/80)

0	entry-type (27)	nbr-of-words in-entry	system indicator	reserved	reserved	nbr-of-log entries-in 224-wd-blk
1	channel function	handler function code	logical-MSA number	path-used-by-this-I/O		
2	unit-status	retry count (neg-if retry-failed)	log-info lost-flg	unused	fault-type	
3	access-control-word					
4	channel-chain-pointer-word					
5	EI-status-word					
6 ≈ 10	reserved					
11	CPU/CAU-number/name					
12	IOU/IOAU-name					
13	control-unit-name					
14	unit-name					
15 ≈ 24	reserved					
25	date-and-time-of-log-entry					
26	system-type					
27	run-id					

Table 9-36. I/O Fault Log Entry (1100/80)

0	*entry-type (27)*	*nbr-of-words in-entry*	*buffer release-flag*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wd-blk*
1	*reserved*					
2	*unit-status*	*instruction type*	*log-Info lost-flg*	*non-zero condition code*	*fault-type*	
3	*interrupt-address-word*					
4	*channel-status-word-0*					
5	*channel-status-word-1*					
6	*channel-status-word-2*					
7	*channel-command-word-0*					
8	*channel-command-word-1*					
9	*I/O-path*					
10	*unused*					
11	*CPU-name*					
12	*IOU-name*					
13	*control-unit-name*					
14	*unit-name*					
15	*reserved*					
24						
25	*date-and-time-of-log-entry*					
26	*system-type*					
27	*run-id*					

where:

fault-type	Type	Definition
	001	1110, 1100/40 CAU/IOAU interface parity(1)
	004	1110, 1100/40 IOAU ACR parity(1)
	005	1110, 1100/40 IOAU storage parity(1)
	006	1110, 1100/40 IOAU channel interface parity(1)
	010	EON input data parity(0)
	011	1100/10, 1100/20 output data parity(0)
	012	EON ISI control word parity(0)
	013	EON ESI control word parity(0)
	017	first log buffer-1(0,1,2)
	020	nonsense interrupt(0,1,2)
	020	nonzero condition code from 'CIOIER'(2)
	022	nonzero condition code from 'SRLSUB'(2)
	023	sense failure detected by 'SRLINT'(2)
	024	nonzero condition code from 'SRLUNL'(2)
	025	machine check(2)
	026	interrupt discarded as 'USTATE' greater than zero(0,1,2)
	027	subchannel status errors(2)
	030	timeout(0,1,2)
	031	nonzero condition code(2)
	032	unsolicited interrupt(0,1,2)

NOTES:

(0) CONFIG parameter EON is turned on.
(1) CONFIG parameter ON40 is turned on.
(2) CONFIG parameter ON80 is turned on.

instruction type

000 - SIOF
001 - TIO
002 - TSC
003 - HDV
004 - HCH
007 - LCBR
010 - LTCW

Table 9-37. Processor Fault Log Entry

0	*entry-type 27*	*nbr-of-wrds in-the log-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*reserved*					
2					*fault-type-040*	
3	*processor fault-type*	*CPU-nbr*				

Table 9-37. Processor Fault Log Entry (continued)

4	*CPU-name*
5	*status 1*
6	*status 2*
7	
≈	≈
24	
25	*date-and-time-of-log-entry*
26	*reserved*
27	*EXEC8*

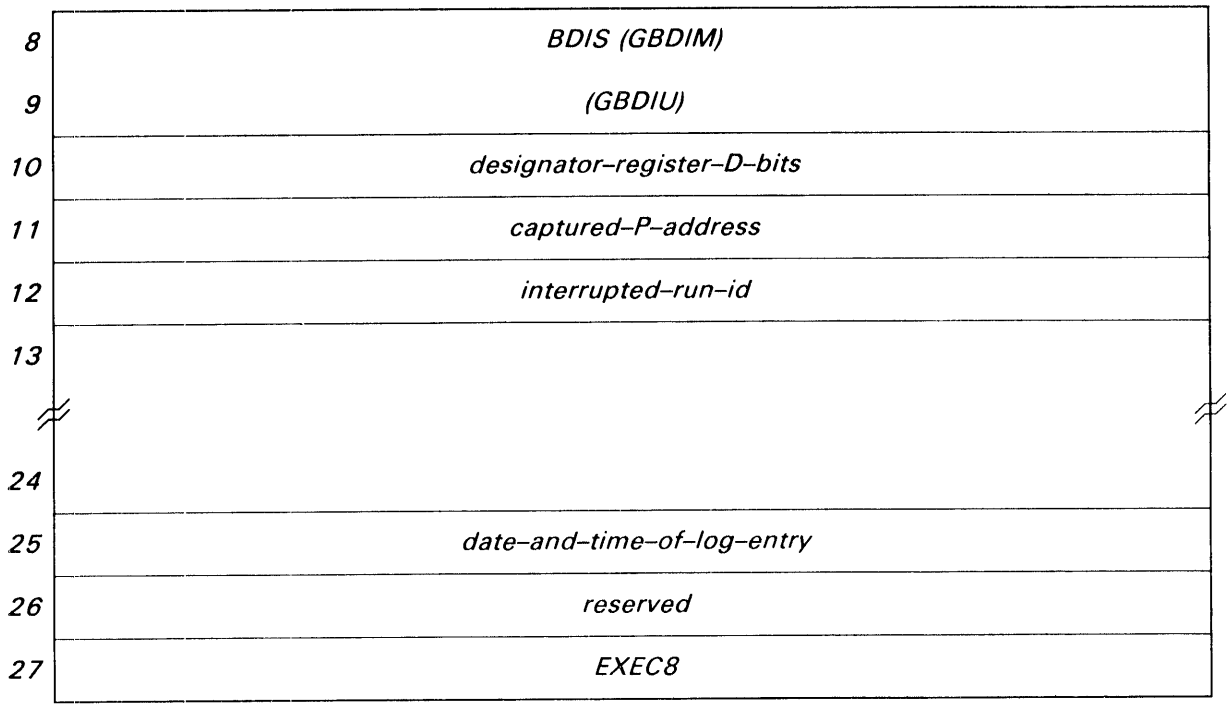
where:

processor fault type
001 PROM Parity Successful Retry
002 PROM Parity Failure
003 MDA Instruction Failure
004 MDB Instruction Failure

Table 9-38. 1100/80 Storage Fault Log Entry

0	*entry-type 27*	*nbr-of-wrds in-the log-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*reserved*					
2				*flg 1*	*fault-type-041*	
3	*flg*	*storage fault-type*	*CPU-nbr*			
4	*MSU-name*					
5	*NSU-relative-address*					
6	*SIU-name*					
7	*status-word*					

Table 9-38. 1100/80 Storage Fault Log Entry (continued)



Word 2

fault type — The storage fault is characterized by the presence of code 041.

flg 1
- 01 true retry
- 02 simulated retry
- 03 no retry permitted

Word 3

flg
- 35 processing complete
- 34 successful retry
- 33 retry failure
- 32 error caused reboot
- 31 count against threshold
- 30 log flag

storage fault type
- 000 ISC – Address or Control Parity Mode
- 001 ISC – Control Parity Error
- 002 ISC – Address Parity Error
- 003 ISC – Write Data Check
- 004 ISC – Read Data Check
- 005 ISC – Address Not Available
- 006 DSC – Tag Parity Error
- 007 DSC – Age Check
- 010 DSC – LRU Check
- 011 DSC – Storage Offline
- 012 DSC – Write Data

013 DSC – Write Address Parity Error
014 DSC – Read Address Parity Error
015 DSC – Read Data
016 DSC – Storage Correctible (Single Bit)
017 DSC – Storage Uncorrectible (Multiple Bit)
020 ISC – Illegal Status
021 DSC – Illegal Status
022 DOWN – CPU
023 DOWN – Block
024 DOWN – MSU
025 DOWN – SIU HALF
026 DOWN – 32K BLOCK
027 DOWN – 64 Word Block
030 UP – CPU
031 UP – Block
032 UP – MSU
033 UP – SIU HALF

Table 9-39. CAU Related Hardware Fault Log Entry (Non-1100/80)

0	*entry-type (27)*	*nbr-of-words in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wd-blk*
1	*status-word*					
2	*retry-cnt (neg-if retry-failed)*	*CAU-nbr*	*IOAU-nbr*	*buffer overlay-flg*	*fault-type 0-CAU/storage-parity-check 1-CAU/GRS-control register-parity-check*	
3	*PSRM*					
4	*PSRME*					
5	*PSRU*					
6	*PSRUE*					
7	*contents-of-P (CAU/GRS-parity-only)*					
8	*contents-of-P+1 (CAU/GRS-parity-only)*					
9	*contents-of-P+2 (CAU/GRS-parity-only)*					
10	*captured-P (CAU/IOAU-interface-parity-only)*					
11	*first-error-address (CAU-storage-parity-only)*					
12	*last-error-address (CAU-storage-parity-only)*					
13	*logical-'AND'-of-erroring-address (CAU-storage-parity-only)*					

Table 9-39. CAU Related Hardware Fault Log Entry (Non-1100/80) (continued)

14	*error-count (CAU-storage-parity-only)*
15	*reserved*
≈	≈
24	
25	*date-and-time-of-log-entry*
26	*reserved*
27	*run-id*

NOTE:

Words 3 through 6 are applicable to CAU/GRS or CAU/storage parity only.

9.1.6.28. Common Bank Reload Log Entry

Whenever a common bank is reloaded, a log entry is created. The request type describes why the common bank was reloaded. The format of this log entry is shown in Table 9-40 and described below.

Table 9-40. Common Bank Reload Log Entry

0	*entry type (28)*	*nbr-of-wds in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wd-blk*
1	*bank-name*					
2						
3				*request-type*		
4						
≈						≈
24						
25	*date-and-time-of-log-entry*					
26	*reserved*					
27	*run-id*					

Word 3

request-type	0001	immediate request (ER)
	0002	error request (ER)
	0004	stall request (ER) register input RL BDI based
	0010	stall request (ER) buffer input RL BDI based
	0020	stall request (ER) register input RL BDI not based
	0040	stall request (ER) buffer input RL BDI not based
	0100	immediate request (console)
	0200	error request (console)
	0400	stall request (console)

9.1.6.29. Log Entry Created by ER LOG$

This log entry is created by the user via ER LOG$ for insertion in the SYS$*SYSTEM$LOG file. The format is shown in Table 9-41.

Table 9-41. User Formatted Log Entry

0	*entry-type (32)*	*nbr-of-words in-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*user-information a-maximum-of-23-words-allowed*					
23						
24	*original-run-id*					
25	*date-and-time-of-entry*					
26	*reserved*					
27	*run-id*					

9.1.6.30. Checkpoint/Restart Termination

Table 9-42. Checkpoint/Restart Termination Entry

0	*entry-type (33)*	*nbr-of-wrds in-the log-entry*	*system indicator*	*reserved*	*reserved*	*nbr-of-log entries-in 224-wrd-blk*
1	*primary-storage-reference-count*					
2	*primary-storage-reference-count*					
3	*extended-storage-reference-count*					
4	*extended-storage-reference-count*					
5	*ER-and-control-card-charge*					
6	*I/O-SUPS*					
≈						
15						
16	*voluntary-delay-time*					
17	*CPU-time*					
18						
≈						
24						
25	*date-and-time-of-last-entry*					
26	*reserved*					
27	*run-id*					

9.2. THE MASTER LOG EDITING PROGRAM (LOGFED)

LOGFED is a user program that reads the master log file, edits and prints the information contained therein.

9.2.1. Calling LOGFED

As a Processor Call

@LOGFED,options

<Field A> <Field B> <Field C>

Alternately

@XQT,options LOGFED

<Field A> <Field B> <Field C>

9.2.2. Parameters

9.2.2.1. Options Field

There are three mutually exclusive options of LOGFED, which may be omitted. They are as follows:

A The master log will be searched for a specified account number, and those log entries with that account number will be printed in edited format.

R The master log will be searched for a specified run-id, and those log entries for that run-id will be printed in edited format.

Z The entire master log will be printed in edited format. For the relative F-cycles specified on the data card.

If more than one option is present, they will take precedence in the order Z, A, R. If no options are present, the parameters on the data card will be unmodified.

9.2.2.2. The Data Image

The image following the program call is the data image. This image provides LOGFED with the parameters necessary to search the master log file. It may consist of three fields, which are dependent upon the option used.

Field A is used only with the 'A' or 'R' options. The run-id or account number specified for the search is placed in this field, starting in image column 1. If not present, image column 1 contains a blank.

Field B contains the type of log entry that is to be edited from the master log file. Multiple entries are separated by commas.

The R option can be used in Field B. This would be the equivalent of entering the following:

1, 2, 3, 4,..., 25, 26, 27, 28 in Field B.

If used, no other parameters are assumed in Field B, and LOGFED will go on to Field C. Use of the R option does not change the meaning of Fields A or C.

Field C denotes the file cycles that the user wishes to edit. Each F-cycle must be separated by a comma. For example, to edit relative F-cycles, -0: -1: -2: -3: The following would be entered in Field C:

```
-0,-1,-2,-3
```

The following would accomplish the same for all log entry types:

Example:

```
@RUN                RUNID,ACCTNO,PROJ,2,100
@ASG,T              TAPE,T,1234C
@COPIN              TAPE,TPF$
@LOGFED,R
SYS R -0,-1,-2
@FIN
```

If any fields are not to be included, such as Field A and Field B when the Z option is specified, then a space is required for positioning. In this case, the first file cycle directive should start in column 3.

9.2.3. Contingency Errors

If an error occurs, LOGFED will print:

'CONTINGENCY ERROR IN LOGFED DUMP FOLLOWS'

LOGFED will then dump all its registers, I-Bank and D-Bank.

The following messages may also be displayed:

INCORRECT PARAMETER CARD

The user's data card is mispunched, LOGFED exits through ERR$.

END OF CYCLE (*-n*)

The end of the cycle has been reached.

SYS*SYSTEM$LOG*(-n)* COULD NOT BE ASSIGNED.
FACILITY STATUS RETURNED: *Oxxxxxxxxxxxx*

where *Oxxxxxxxxxxxx* is the status received from the attempted assignment.

I/O STATUS RETURNED *xxx* SWITCHING TO NEXT F-CYCLE.

where *xxx* is the status returned from I/O.

LOGFED, prior to printing this message, will dump all registers and the I/O packet.

9.2.4. LOGFED Features

9.2.4.1. Specification of Subcodes

The caller of LOGFED may specify on the parameter card which subcodes of the type 24 (software-detected error) log entries are to be edited. The subcodes (1 thru 72) are listed following the type 24 specification, separated by a slash (/) character, e.g., 5,17,24/3/4/5,25,26. If no subcodes are specified following the type 24 specification, then all subcodes are edited.

9.2.4.2. Swap Tapes for Multi-Reel Files

When the system detects an end-of-reel for the log file, it writes back-to-back EOFs to close a log file F-cycle on tape. The same sequence is written by the Executive on an auto-reboot to close the current log file F-cycle, if so configured. The end-of-reel for a log file F-cycle continued on the next tape is indicated by one EOF mark, followed by a 14-word block, followed by two EOFs. The first word of the 14-word block contains an SDF end-of-reel code (0541600000000). LOGFED recognizes this "continuation" sentinel and swaps to the next tape reel.

9.2.4.3. Invalid Log Entries

Upon encountering a log entry type outside of the valid range, LOGFED will edit all words in the log entry, except for time, date, and run-id, in 12-character octal notation, due to the unknown format.

9.3. STANDARD UNIT OF PROCESSING

9.3.1. Introduction

This subsection describes the concept of SUPs and answers some of the common questions concerning SUPs. The topics discussed are:

- What are SUPs?
- Why are they computed that way?
- How do SUPs relate to system performance comparisons?
- What does the EXEC do with SUPs internally?
- How should SUPs be used for accounting and billing?

9.3.2. Description

The Standard Unit of Processing (SUP) is used as a processing time measure. Processing time includes all time duration oriented services provided by the system. These are CPU (CAU) utilization, I/O channel utilization, and fixed charges representative of the previous two for control statement (CC) and ER processing performed by the Executive at user request.

The formula for SUPs is:

SUPs = CPU SUPs + ER/CC SUPs + I/O SUPs

9.3.3. Computation

Accumulations occur in a consistent manner regardless of order of processing and actual time durations and sequences. SUPs thus truly represent a standard unit of processing. A program that performs exactly the same operations each time it is run should get the same charges each time. There will be some small (usually less than one percent) variations due to some inaccuracies in the CPU time measurement. For 1110 and 1100/40 Systems this inaccuracy is very small due to the use of the storage reference counters (SRCs) for CPU usage measurements.

On 1108, 1106, 1100/10 and 1100/20 systems a somewhat larger inaccuracy results from the use of the real time clock to measure CPU usage. The measured time is in CPU SUPs and is in 200 microsecond units. ER and I/O SUPs should be truly identical unless a general equipment type was requested for a file or the facilities configuration changed and different allocation resulted.

The intent of defining a Standard Unit of Processing is twofold. First, a single number indicative of system utilization by a run is needed for many reasons including time sharing, max time checks and deadline time scheduling. Secondly, a reproducible number is needed for the above reasons and for accounting. SUPs provide most of these requirements and when combined with main storage utilization to form Core Block SUPs (CBSUPs) provide an excellent value for all of these cases.

CPU/CAU SUPs are the charge that the Executive records for the use of the central processing unit(s) by the user program. This charge is calculated differently for each basic machine type in the 1100 Series family.

9.3.3.1. CPU SUPs (1108, 1106, 1100/10 and 1100/20)

The real time clock is used to measure the user's CPU time. This measured time is used as the CPU SUPs and is in 200 microsecond units.

9.3.3.2. CAU SUPs (1110 and 1100/40)

The storage reference counts are used to calculate the CAU SUPs for this type of machine. The formula is as follows:

$$\text{CAU SUPs} = (SRC_p + SRC_E) * K_1$$

where:

SRC_p number of primary storage references

SRC_E number of extended storage references

K_1 $\quad \frac{700}{1.6*200000} = 2.19 \times 10^{-3}$

700 average instruction execution time in nanoseconds

1.6 number of storage references per instruction

200000 convert to 200 μsecond increments

CAU SUPs is in 200 μseconds units

9.3.3.3. CPU SUPs (1100/80)

The Quantum Timer is used to accumulate the CPU SUPs for this type of machine. Its granularity is 100 nanoseconds. The accumulated time is converted to 200 microsecond increments to produce CPU SUPs.

$$\text{CPU SUPs} = \frac{\text{Quantum Timer time}}{2000}$$

The Quantum timer does not run during SIU wait time due to SIU misses, conflicts at SIU ports, or SIU service delays. Thus it is only an accounting measure of CPU usage, not CPU–SIU usage. CPU SUPS must not be taken as direct performance measures of the CPU–SIU resource usage, although they can be appropriately biased according to particular site hardware and software characteristics to yield accurate performance measurements.

9.3.3.4. Control Statement and Executive Request SUP Charges

Control statement (CC) and Executive Request (ER) SUP charges are a charge that the Executive records for all types of requests made to the Executive. In general this charge represents the work that the Executive must do in order to satisfy the user request. This work is usually the instructions executed by the Executive but occasionally may include I/O time if the I/O time cannot be charged to the user as I/O SUPs.

9.3.3.4.1. Control Statement Charges

Currently, the Control Statement SUPs are expressed as the number of 200 μsec increments that will be added to the users ER/Control Statement charges whenever a Control Statement is encountered either in a runstream or through a CSF$ request. All ERs executed by the Executive code which is servicing the Control Statement will be charged to the user as additional ER/CC charges. Also, most I/O transfers performed by the Executive Control Statement service routine will be charged to the user as I/O SUPs.

The following is the list of charges for control statements.

Control Statement	SUP Charge	Control Statement	SUP Charge
@ADD	3300	@MODE	70
@ASG	3300	@MSG	30
@BRKPT	6500	@PASSWD	500
@CAT	2500	@QUAL	50
@CKPT	225	@RSTRT	4500
@END	15	@RUN	6500
@ENDF	10	@SETC	30
@EOF	10	@START	4300
@FILE	4500	@SYM	1500
@FIN	6500	@TEST	40
@FREE	2000	@USE	70
@HDG	30	@XQT	375
@JUMP	90	Processor Call	375
@LOG	80		

9.3.3.4.2. Executive Request Charges

The ER charge is stored internal to the Executive as an approximation of the number of Executive instructions necessary to perform the requested service. There is no guarantee that these numbers are accurate. At system generation time the instruction count is converted to 200 μsec increments by a calculation similar to the following:

$$\text{ER SUPs} = \frac{\text{INST}}{K_3}$$

where:

INST is the number of instructions

K_3 is a constant for a particular type of 1100 Series computer. Each 1100 Series has a unique value. The formula for K_3 is:

$$K_3 = \frac{200000}{IT}$$

200000 converts to the number of μsecond increments
IT is the instruction time for the particular 1100 Series computer.

The following is the table of values for K_3 for the 1100 Series computers.

Machine	IT	K_3
1100/80	525	381
1100/40	700	286
1100/20	1192	168
1100/10	1468	136
1110	700	286
1108	1100	182
1106II	1600	125
1106I	2200	91

NOTE:

The IT (instruction time) values used are only the values used by the Executive and are not meant to be the actual performance of the machine. In most cases they are reasonable approximatations.

ERs to CSF$ receive a charge for the ERs as well as a charge for the Control Statement that is being processed. All ERs executed by the ER service code will be charged to the user as additional ER/CC SUP charges. Also, most I/O transfers performed by the Executive ER service routine will be charged to the user as I/O SUPs.

The costs for all of the ERs are shown as the number of instructions for each ER and then the formulas from above are applied to calculate the number of 200 μsec increments for the fastest (1100/80) and slowest (1106I) machines.

9.3.3.4.3. Standard Executive Requests

The following are the list of charges for standard Executive Requests.

Executive Request	Function Code	Approximate Nbr-of-Inst	1100/80	1106I
ABORT$	012	2000	5	21
ABSAD$	030	2000	5	21
ACCNT$	0163	1500	3	16
ACSF$	0140	2000	5	21
ACT$	0147	500	1	5
ADED$	0161	300	1	3
APCHCA$	077	3000	7	32
APCHCN$	075	3000	7	32
APNCHA$	073	1000	2	10
APRINT$	070	1000	2	10
APRNTA$	071	1000	2	10
APRTCA$	076	3000	7	32
APRTCN$	074	3000	7	32
APUNCH$	072	1000	2	10
AREAD$	0166	1500	3	16
AREADA$	0167	1500	3	16
ATREAD$	0170	3000	7	32
AWAIT$	0134	1000	2	10
BANK$	0160	1500	3	16

Executive Request	Function Code	Approximate Nbr-of-Inst	1100/80	1106I
BEOF$	036	1400	3	15
BDSPT$	0115	4600	12	50
CADD$	057	1000	2	10
CEND$	0100	50	1	1
CGET$	056	1000	2	10
CJOIN$	0151	2000	5	21
CKRS$	0204	3000	7	32
CLIST$	0153	1500	3	16
CMD$	051	2000	5	21
CMH$	052	2000	5	21
CMI$	047	1400	3	15
CMO$	050	1400	3	15
CMS$	045	2000	5	21
CMSA$	053	2200	5	24
CMT$	046	2000	5	21
COM$	010	1600	4	17
COND$	066	90	1	1
CPOOL$	055	2300	6	25
CQUE$	0117	2000	5	21
CREL$	0152	2000	5	21
CRTN$	035	90	1	1
CSF$	017	2000	5	21
CTS$	0123	300	1	3
CTSA$	0124	300	1	3
CTSQ$	0122	400	1	4
DACT$	0150	550	1	6
DATE$	022	132	1	1
EABT$	026	2000	5	15
EDJS$	04	1500	3	16
ERR$	040	2000	5	15
ERRPR$	0202	3000	7	32
EXIT$	011	850	2	9
FACIL$	0114	1400	3	15
FACIT$	0143	1400	3	15
FITEM$	032	1400	3	15
FORK$	013	750	1	8
IALL$	0101	300	1	3
IDENT$	034	100	1	1
II$	027	2500	6	27
INFO$	0116	1400	3	15
INT$	033	200	1	2
IO$	01	2100	5	23
IOADH$	0206	2800	7	30
IOARB$	021	2700	7	29
IOAXI$	020	2700	7	29
IOI$	02	2700	7	29
IOW$	03	2200	5	24
IOWI$	024	2700	7	29
IOXI$	025	2700	7	29
LABEL$	031	2000	5	21
LCORE$	042	4100	10	45
LOAD$	0111	7300	19	80

Executive Request	Function Code	Approximate Nbr-of-Inst	1100/80	1106I
LOG$	0210	2000	5	16
MCORE$	043	11200	29	123
MCT$	041	9600	25	105
MSCON$	0125	15000	39	164
NAME$	0146	1500	3	16
NRT$	062	100	1	1
OPT$	063	90	1	1
PCHCA$	0165	3000	7	32
PCHCN$	0164	3000	7	32
PCT$	064	400	1	4
PFD$	0106	1500	3	16
PFI$	0104	1500	3	16
PFS$	0105	1500	3	16
PFUWL$	0107	1500	3	16
PFWL$	0110	1500	3	16
PNCHA$	0145	1000	2	10
PRINT$	016	1000	2	10
PRNTA$	0144	1000	2	10
PRTCA$	0155	3000	7	32
PRTCN$	0137	3000	7	32
PSR$	0157	125	1	1
PUNCH$	0130	1000	2	10
READ$	015	1000	2	10
READA$	042	1000	2	10
ROUTE$	0133	1500	3	16
RSI$	0112	2000	5	21
RT$	061	100	1	1
SETBP$	0156	1500	3	—
SETC$	065	90	1	1
SMU$	0211	1500	3	16
SNAP$	0126	3000	7	32
SWAIT$	0103	800	2	8
SWTCH$	0127	200	1	2
SYMB$	0200	2000	5	16
SYSBAL$	0176	5000	13	54
TDATE$	054	100	1	1
TFORK$	014	1500	3	16
TIME$	023	100	1	1
TINTL$	0136	3000	7	32
TLBL$	0142	2000	5	21
TREAD$	0102	2000	5	21
TRMRG$	0120	2000	5	21
TSQCL$	0113	90	1	1
TSQRG$	0121	90	1	1
TSWAP$	0135	3000	7	32
TWAIT$	060	800	2	8
UNLCK$	067	800	2	8
USER$	0177	2000	5	16
WAIT$	06	850	2	9
WALL$	037	800	2	8
WANY$	07	850	2	9
XCTS	0162	1500	3	16

9.3.3.4.4. TIP Executive Requests

The following are the TIP Executive Requests:

TIP ER	Function Code	Approximate Nbr-of-Inst	1100/80	1106I
AC$NIT	02030	2000	5	21
BT$ENA	02055	100	1	1
BT$DIS	02054	100	1	1
CA$ASG	02011	170	1	1
CA$REL	02012	180	1	1
CR$ELG	02021	200	1	2
CR$LGR	02020	170	1	1
CR$LOG	02017	170	1	1
CR$PHS	02014	100	1	1
CR$VAL	02032	2000	5	21
CS$LOG	02015	170	1	1
CS$OUR	02016	170	1	1
CS$PHS	02013	260	1	2
DM$FAC	02050	800	2	8
DM$IO	02051	2000	5	21
DM$IOW	02052	2000	5	21
DM$WT	02053	100	1	1
FCEG	02045	1500	3	16
FC$SSN	02043	2000	5	21
FL$BOX	02056	100	1	1
ME$GET	02063	1500	3	16
PB$CON	02002	1500	3	16
PB$DIS	02003	1500	3	16
PI$NIT	02000	450	1	4
RT$INT	02004	1500	3	16
RT$OUT	02005	300	1	3
RT$PSD	02061	5000	13	54
RT$PSI	02060	5000	13	54
RT$SCH	02040	300	1	3
RT$TRS	02007	200	1	2
TPFLG$	02062	1500	3	16
TPLOG$	02033	1900	4	20
TRTIM$	02057	100	1	1
UK$ONS	02034	250	1	2
UO$AND	02035	1500	3	16
UO$GET	02036	1500	3	16
UO$OR	02037	1500	3	16
UT$DEL	02026	1500	3	16
UT$EAQ	02022	1500	3	16
UT$ERQ	02023	1500	3	16
UT$UPA	02024	1500	3	16
UT$UPR	02025	1500	3	16
UT$TIM	02027	1500	3	16

9.3.3.5. I/O SUPs

I/O SUPs are the charge that the Executive records for all input/output of data. In general this charge represents the time that the channels and devices attached to the system are busy.

9.3.3.5.1. Internal Executive Formula

The formula used by the Executive is as follows:

$$\text{I/O SUPs} = \sum_{i=1}^{G} R_i * A_i + W_i * T_i$$

where:

R — is the number of requests or accesses for an I/O group.

W — is the number of words transferred for an I/O group.

G — is the configured number of I/O groups in the system.

A — is the configured access time of the I/O group.

T — is the configured time to transfer one word for this I/O group.

Items G, A, and T are configuration parameters for the Executive.

In all cases the accumulations are made for the service requested and not necessarily for the services provided. For example, if a user requests that a file be allocated to an FH432 drum, but the Executive actually placed it on an 8430 disk, the A and T parameters used to calculate the I/O SUPs for that file will be the FH432 values. However, if a general request is made (e.g. F) or the requested facilities are not configured as part of the system, the A and T parameters will be those of the first device group actually allocated.

Generally all I/O by the Executive to files assigned or created by the user program is charged to the user as I/O SUPS. Some examples of Executive I/O requests that will cause user I/O SUPs to accumulate follow.

All symbiont requests (e.g. PRINT$, READ$, SNAP$, etc.)
User program initial load
User program segment load
Writing user programs to the diagnostic file
MSCON$ request which writes into the user file
All program file requests (e.g. PFI$, PFS$ etc.)
Checkpoint/Restart I/O operations

9.3.3.5.2. System Log File Formula

Because the Executive's log file does not capture both R and W, (see 9.3.3.5.1) a different formula must be used whenever I/O SUPs are calculated by a log file analysis routine. The log file data has only one piece of data for every I/O group and it is defined as follows:

$$\text{I/O Group}_i = \frac{A_i}{T_i} * R_i + W$$

The formula to calculate I/O SUPs from the log file is:

$$\text{I/O SUPs} = \sum_{i=1}^{G} \text{I/O Group}_i * T_i$$

9.3.4. Relation to System Performance Comparisons

The first and most important problem with respect to performance comparisons is to get a definition of what is meant in each case. The second problem is that of determining what aspect of performance is most important to the site.

Three aspects of performance are:

1. The number of jobs per unit time that can get done. This may also be stated as the amount of time (wall time) it takes to complete a certain mix of jobs.

2. The number of dollars that can be collected from clients based on the billing parameters provided by the system.

3. The maximum and/or average response time for all transactions.

In choosing the aspect of performance to examine most heavily, one must also consider what differences exist between the systems to be compared.

If there is any kind of major hardware difference between the systems, the system-provided billing parameters are of little or no use. The only important measurement in such a case is an actual benchmark that is as representative as possible of the actual work to be done on the system. Not only can one be easily misled by system accumulated data but, until actual experience is obtained on the system, the relationships between this data and useful work cannot be determined.

Once a benchmark is established, it is probably easiest to compare aspect 3 of performance. Establishing such a benchmark may be very difficult due to differences between systems and the problems of creating adequate transaction drives.

Comparisons of aspect 2 of performance are not meaningful in terms of benchmarks. The most that can be accomplished is to determine whether or not a system supplies meaningful billing data.

Comparisons of performance as described above aspect 1 of should be achieved by running the same set(s) of jobs on the systems and noting the average number of jobs completing in a sufficiently long time period (e.g., 24 hours). Obviously, this entails a lot of work. The idea is to eliminate the starting and ending transient conditions that exist in any system and to include both short and long running jobs. The time period required is determined by the size of the jobs to be run and should be one or two orders of magnitude greater than the time required to run the average job. An approximation to this can be achieved by running a benchmark that repeats the same set of jobs a sufficient number of times to provide a reasonable period during which the system is running at full capacity. The completion rate is then determined for this time period.

None of this is intended to mean that the starting and ending transient periods are totally without effect on overall system performance. If a system is not used constantly, the total work accomplished in one operational period is obviously affected by these time constants. For short shift operation,

abnormally long times may measurably affect total performance. However, this affect can also be determined by the above benchmark.

SUPs are of no utility in comparing different systems, but may be very useful when comparing small variations in configuration and work load on a given system. The CBSUPs accumulated per unit time is probably the best single measure of useful work accomplished that is presented by the EXEC. When combined with response time data from SIP, it gives a very accurate comparison between the two time periods. It should be noted however, that CBSUPs and response time are of little use in diagnosing the reasons for changes.

9.3.5. Uses by the EXEC

SUPs are used for four primary purposes by the EXEC. In some cases, SUPs are used by themselves to provide a measure of virtual time towards completion as seen by the run. In other cases, SUPs are combined with other parameters to provide a measure of system resource utilization.

1. SUPs are used to prevent unplanned loops in a program from causing it to run indefinitely. SUPs, then, are the units described by the max time field on the RUN control statement. In previous levels (27 and before) of the Executive, this field contained units of CPU time. However, it was noted by many sites that runs with ER, I/O, or print loops could loop for a very long time before the run was terminated. The usage of SUPs ensures that all time dependent resources are included in the determination and that all possible loops will be detected.

2. SUPs are used to control deadline scheduling. The deadline scheduling algorithm is based on the following formula:

$$K = \frac{\text{deadline time} - \text{current time}}{\text{max time SUPs} - \text{used SUPs}}$$

   The value of K is then compared to a series of constants to determine whether the run must be treated as critical deadline or not, and, if it is, at what deadline priority it should be scheduled. For unopened runs there are six deadline priorities (0 thru 5) that represent the criticality of the run. Note that 'X' option (e.g., ROLOUT, ROLBAK) runs also use priority zero. A priority 'A' run is at priority six by comparison. If the value of K transforms into the range zero through five, the run is critical deadline. If K transforms into a number greater than five, the normal run priority is applied. The standard transformation is to use K exactly as given by the above formula. Once a run is opened it is only deadline or non-deadline. The above computation is performed approximately once per minute in both cases.

3. SUPs combined with main storage usage to form Core Block SUPs (CBSUPs) are used to drive the time sharing and demand/batch sharing algorithms. The combination is multiplicative and is performed in steps so that the result is as close as possible to an integral of storage size times SUPs. This number gives the best single measure of the impact on the system of running that program. CBSUPs is the number that is used to determine when a program's main storage quantum for time sharing has been exceeded. It is also accumulated for all demand and all batch programs in order to control demand/batch sharing.

   The batch, demand, and real time usage values displayed on the continuous display portion of the console are also based on CBSUPs. A maximum theoretical CBSUPs value is computed by multiplying the amount of user main storage by 12 seconds worth of SUPs. This gives the number of CBSUPs that would be accumulated in 12 seconds if all of the storage was filled by a single continuously executing program. This could also be achieved by having several programs that were all in some form of execution (CPU or I/O) simultaneously. On an 1110 or 1100/40 System, all computation would have to occur in primary storage with only I/O occurring in extended storage in order to reach this value. Every six seconds the accumulated

CBSUPs for each of the three program types is added to the corresponding value from the previous six seconds and then divided by the theoretical value. This result expressed as a percentage is what is displayed on the console.

4. SUPs and CBSUPs are among the parameters that are logged for accounting purposes. SUPs are also combined with voluntary wait time and integrated with mass storage space to compute the track seconds that are logged for temporary file usage.

9.3.6. Accounting and Billing Uses of SUPs

SUPs and CBSUPs provide a useful base for determining charges to be applied. Obviously, billing is in large part determined by the dollar costs attached to each parameter. However, this is not enough. The billing parameters and their assigned costs must represent the actual effect on the total system of running that particular job. The results must also be repeatable so that the same job accumulates the same charges (exclusive of rate variations) each time it runs. However, a job which is run twice may validly accumulate different charges for several reasons. It may have timing dependent code such as instruction loops to wait for I/O completion or the usage of Test and Set instructions to synchronize multiple activities. Or, a program may be designed to request some facility and, if it is not immediately available, a higher or lower performance (price) facility will be acquired.

The EXEC attempts to provide as much information as possible about all the system resources that a program (run) utilizes. This information is written to the master log file. Billing programs are expected to scan the log to obtain charging information. There is, however, a much reduced and incomplete summary that is kept in the summary account file.

In order to set up a billing system, a careful analysis of the system must be performed. Data on all billing parameters should be accumulated for a representative period (e.g., one month). From this data it can be determined what the critical resources of the system are. In most cases it will be desirable to use SIP to better isolate critical resources (bottlenecks). The billing parameters cover all aspects of the system hardware including mass storage utilization, I/O channel utilization, number of images read, printed and punched, and CPU time.

This data should then be used along with the normal financial determinants of system and component cost, profit, competition, etc. to determine the charges to be applied to each parameter. It should be noted that this must be an on-going procedure. System utilization will change, hardware configuration will change and the Operating System software will change. All of these items and more will have an effect on the charges and/or return.

9.3.7. Conclusion

SUPs, and especially CBSUPs, provide meaningful data for measuring the impact on the system of running a specific program or run. This data includes some reflection of all the resources utilized by the run and as such provides a convenient basis for controlling system utilization.

USER COMMENT SHEET

Comments concerning the content, style, and usefulness of this manual may be made in the space provided below. Please fill in the requested information.

Requests for copies of manuals, lists of manuals, pricing information, etc. should be made through your 1100 Series site manager to your Sperry Univac representative or the Sperry Univac office serving your locality.

System: _____

Manual Title: _____

UP No: _____ Revision No: _____ Update: _____

Name of User: _____

Address of User: _____

Comments:

CUT

FOLD

FIRST CLASS

PERMIT NO. 21

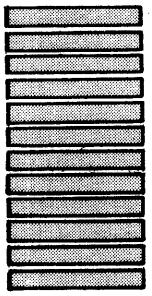
BLUE BELL, PA.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

SPERRY UNIVAC



CUT

SYSTEMS SUPPORT
ATTN: INFORMATION SERVICES M.S. 4533
P.O. BOX 3942
ST. PAUL, MINNESOTA 55165

FOLD