File No. S370-36
Order No. GC28-2008-5

Systems

IBM Time Sharing System System Programmer's Guide

IBM Time Sharing System (TSS) makes a distinction between user and system programmers. This publication is intended for persons responsible for maintaining, modifying, or extending the system and discusses:

- Operating environment
- Program structure
- Coding practices and conventions
- Privileged supervisor call instructions
- Serviceability aids
- System macro definitions
- Changing TSS
- Privilege Class E

Sixth Edition (July 1979)

This edition revises and makes obsolete GC28-2008-4. Among the modifications made to TSS that are reflected in this edition are the following:

- Macros have been updated to reflect new operands, new examples of use, etc., and editorial corrections have been made.
- Complete descriptions of the following macros have been added:

  ESEG EXCSEG MAPTDY RTTCTL

- Special operands available only to the systems programmer have been added for the following macros:

  DISCSEG NIB PR RSVSEG

- Complete descriptions of the following system programmer type commands have been added:

  ALDPOOL BLDSVCT CNVTPOOL DISP DUMPRES GTF POOL? TRACE
  BLDPOOL CLASSGTF DELPOOL DSCBS FIXCAT MOVEUSER SETRVN TRACEND

- Descriptions have been added for the following: Deadline Dispatcher; Generalized Trace Facility; Public (Storage) Pools; and Network Control Program (NCP) Support. Also, the System Enter Code, Virtual & Real Memory SVC, and Extended Program Interruption Code tables have been updated, and RTAM Log Entry Definitions have been added.

This edition reflects system changes made by PRPQ 5799 AXA and Release 3.09 of the IBM Time Sharing System/370 (TSS/370), and remains in effect for all subsequent versions or modifications of TSS unless otherwise noted. Significant changes or additions to this publication will be provided in new editions or Technical Newsletters.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to: IBM Corporation, Time Sharing System, Dept 80M, 1133 Westchester Avenue, White Plains, New York 10604.

PREFACE

This publication will aid you -- as a system programmer -- to extend and modify the IBM Time Sharing System. Available programming facilities and necessary coding conventions are described. Examples are provided to give you an understanding of what is involved in changing the system.

Part I contains both orientation and how-to information and is divided into:

- Section 1: An introduction to system programming and system programmer's facilities.
- Section 2: The basic concepts and structure of TSS.
- Section 3: Conventions to use when coding routines for TSS.
- Section 4: Facilities available to system programmers; information on how they can be used and, in some cases, changed.
- Section 5: How to write macro definitions for use in the system.

Part II provides a reference to macro instructions (not described in Assembler User Macro Instructions) available to system programmers (and several options available only to system programmers), and is divided into:

- Section 1: How macro instructions are described.
- Section 2: Descriptions of the macro instructions, arranged alphabetically.

Part III discusses commands that are available only to the system programmer, or that have special options available only to the system programmer.

READER'S GUIDE

Several other TSS publications contain information related to system programming and monitoring.

- Concepts and Facilities, GC28-2003, presents the basic concepts and features of TSS.
- System Generation and Maintenance, GC28-2010, describes the procedure for creating and maintaining the source and object forms of TSS; specifically, the macro instructions and commands you may use to add, delete, or modify system object modules.
- Time Sharing Support System, GC28-2006, describes the on-line program error analysis system designed specifically for system programmers.
- Multiterminal-Task Programming and Operation, GC28-2034, describes multiterminal task programming techniques. You should, of course, be familiar with other IBM Time Sharing System publications, such as:

Assembler Programmer's Guide,GC28-2032

Assembler User Macro Instructions, GC28-2004

CONTENTS

This part describes a system programmer, his functions, the system facilities available to him, and the system program structures and conventions that he must observe when coding system modules.

SECTION 1: INTRODUCTION

Time Sharing System (TSS) is a set of programs. Each program performs a part of the overall job that the system as a whole was designed and developed to do. System programming with TSS involves adding to, deleting from, or modifying these programs.

What a System Programmer Does

As a system programmer, you are expected to be an experienced programmer responsible for modifying, extending, and generally adapting TSS to suit the needs of your installation. To do this, you should be knowledgeable in two areas: (1) the design and construction of TSS, and (2) the needs and capacity of your installation.

Why TSS Is a Modular System

Any large, general-purpose programming system is a compromise of the many conflicting demands of its prospective users. System designers attempt to construct an efficient programming system that will satisfy diverse demands. All situations can never be anticipated. Generality must sometimes be sacrificed for efficiency. Realizing this, the developers of TSS have produced a modular system whose parts can be changed. The rules, suggestions, and operating considerations for changing the system are described in the following pages.

THE SYSTEM PROGRAMMER

A programmer becomes known to TSS as a system programmer when he is joined to the system by the system manager or one of the system administrators with a special authority code (O or P).

SYSTEM PROGRAMMER AUTHORITY CODES

The JOIN command contains an authority code which may have the values U (user), P (system programmer), or O (privileged system programmer). As a system programmer, you will have been given an O or P. When a user logs on, information is taken from the user table built by the JOIN command processor and inserted into the user's task status index (TSI) and interrupt storage area (ISA). The SVC queue processor controls what programs are allowed to issue privileged SVCs; it uses the authority code information the LOGON processor stores in the task status index at field (TSIF4) for this purpose. The dynamic loader and program control system use information stored by the LOGON processor in the interruption storage area field (ISAUTH) to determine if the task may perform certain privileged operations.

PRIVILEGE CLASS

As a system programmer, you may be joined to the system with combined privilege classes D and E; each class is associated with a particular set of facilities available for your use. The system programmer with class D and E privilege is often referred to, in TSS publications, as the system monitor. Authority O or P is not, however, a prerequisite of Privilege E.

The assignment of privilege class D (along with your authority code of P or O) designates you as a system programmer. This privilege class provides you with the facilities described in Assembler User Macro Instructions and Command System User's Guide; in conjunction with your authority code, class D also provides you with most of the facilities discussed under "System Programming Facilities" in this publication.

The assignment of privilege class E, which designates you as a system monitor, extends the range of facilities available to you. Through certain options that only the privilege class E programmer can use in the DDEF command and macro instruction, in the DCB macro instruction, and in the SECURE command, you can reserve specific I/O devices and directly utilize unit record equipment. It also provides you with the ability to use the Multiple Sequential Access Mehtod (MSAM), denied to ordinary users and to system programmers who have not been assigned privilege class E.

RESPONSIBILITIES OF A SYSTEM PROGRAMMER

System programmers are responsible for generating the specific version of TSS used at each installation and for troubleshooting and maintaining that system once it is generated. Maintenance involves analyzing system problems, designing changes (additions, deletions, etc.), and incorporating IBM-issued changes applicable to the installation.

System Generation

System generation consists basically of reassembling and replacing system modules containing configuration-dependent tables and installation-option parameters. System generation macro instructions are used to control this operation. These macro instructions, as well as the system generation process, are described in System Generation and Maintenance, GC28-2010.

System Maintenance

You should not attempt to modify TSS unless you have a thorough knowledge of the system's logic, in particular of the interfaces involved in each modification. Detailed information about methods of installing system modifications can be found in System Generation and Maintenance.

SYSTEM PROGRAMMING FACILITIES

A brief summary of the facilities available to system programmers is presented below. The facilities available to system programmers are usually invoked by issuing system-defined macro instructions and commands. Macro instructions are described in detail in Part II of this publication. Commands are described in detail in Part III. General use of these facilities by system programmers is described in Section 4, "System Facilities," in Part I of this publication.

MACRO INSTRUCTIONS

Two groups of macro instructions are provided in TSS. One group is made available to the ordinary user to aid him in managing his data and programs. A second group is provided for system programmers to aid them in the system generation process and in coding system modules.

System Programmer Macro Instructions

Many different macro instructions are available to a system programmer. He may use: the macro instructions provided for generating TSS and adopting it to an installation's requirements, the macro instructions employed within system code, and the macro instructions provided to the ordinary user of TSS, as indicated below under "User Macro Instructions." If he has been joined to the system with both privilege classes D and E, he can also employ macro instructions used by the the Multiple Sequential Access Method (MSAM).

User Macro Instructions: Macro instructions available to all TSS users are described in Assembler User Macro Instructions.

System Generation Macro Instructions: System generation macro instructions perform several basic functions; they inform the software system of the hardware configuration of an installation, they establish the command system options and defaults, and they assign the task management parameters that are used for dispatching and controlling tasks within TSS. These task management parameters are used by TSS to manipulate virtual storage, control paging operations, and regulate the size and number of tasks by type. These macro instructions are described in detail in System Generation and Maintenance.

System Macro Instructions: The system macro instructions used in system programs and those available for the development of complex installation functions are described in detail in Part II of this publication; macros usable only in real memory are called supervisor macros.

Restrictions on Use of System Macro Instructions

Most system macro instructions generate Supervisor Call instructions (SVCs) to establish linkage to a system-provided routine. The dispatching of these system routines is controlled by SVC queue processors. System programmers should be aware that use of many of these SVCs is restricted by the queue processor (and occasionally by the system routine that is called) to privileged users having a certain authority code. SVCs issued in nonprivileged code generally pass control to system programs in privileged virtual storage; those issued in privileged code pass control to system programs in main storage. These are generally referred to, respectively, as nonprivileged and privileged SVCs. Some SVCs can be executed in both nonprivileged and privileged code. A summary of the requirements for assembly and execution of particular SVCs can be found in Appendix B.

COMMANDS

Many TSS commands are available only to system programmers. There are also several commands, available to all TSS users, that have particular options that are available only to system programmers. These commands generally require that a system programmer be joined to the system with authority code O or P, privilege class E, or the userid TSS*****.

The commands defined for system programmers fall into three groups: general services, serviceability aids, and system monitor facilities.

General Services

The general service commands provide system programmers with message maintenance and storage maintenance facilities. Detailed descriptions of these commands are included under "System Programmer Commands" in Part III.

Serviceability Aids

The following commands are for monitoring system performance and for analyzing sources of system errors.

- Program Control System (PCS): A set of commands that enable you to locate problem sources in nonprivileged, virtual storage programs. PCS also provides similar, but restricted, facilities for trouble-shooting privileged virtual storage programs. (No PCS facilities are available for resident programs.) The use of these facilities is discussed briefly in Part I, Section 4. A detailed description of the facilities can be found in Command System User's Guide.

- Time Sharing Support System (TSSS): A support system that enables you to gain access to all storage (real, virtual, and auxiliary) and all registers from a terminal. This support incorporates a command language which lets you dynamically modify both system and user tasks. Like PCS, it is a subsystem within TSS; unlike PCS, it is available only to the system programmer. The use of these facilities is discussed more fully in Part I, Section 4. A detailed description of the facilities can be found in Time Sharing Support System.

System Monitor Facilities

A special set of command options is available for system programmers that have a privilege class of E (system monitor). These options allow the programmer to reserve unit record equipment for nonconversational tasks, refer to devices symbolically, and to print ASCII data sets from tape devices. The special operand values of the SECURE, DDEF, and PRINT commands, that provide the class-E programmer with these capabilities, are described in Part I, Section 4, under "Extended System Monitor Facilities."

TSS ORGANIZATION

The programs that make up TSS are of two types: resident programs, which are brought into main storage and left there until the machine is turned off; and nonresident or virtual storage programs, which are brought into main storage as required and are removed from main storage when the space is needed. During the operation of TSS, both kinds of programs interact by using a well defined interface. This interface is described under "Tasks."

Resident programs schedule the use of the system's resources. They monitor multiprogramming and multiprocessing in TSS. Nonresident programs provide services to the user, making it easier for him to use the system. An attempt has been made to separate these responsibilities as much as possible.

RESIDENT PROGRAMS

This section discusses the characteristics of the TSS resident supervisor. If you are primarily interested in scheduling and resource allocation, you will find this section of special interest.

GETTING STARTED

In TSS, a program called Startup brings into main storage all modules that make up the resident supervisor. Resident programs have the same structure as any other TSS object modules; they have a program module dictionary (PMD) and text. Startup reads the various resident object modules from a disk pack called the IPL volume, resolves the symbolic references between the modules, assigns them main storage, and resolves address constants contained in them. Startup also initializes prefixed storage areas (PSAs) and issues an external start to a second processing unit, if one is attached.

Resident modules initially contain address constants; however, once Startup has transferred control to the resident supervisor, these address constants have been resolved and the relocation of resident programs is complete.

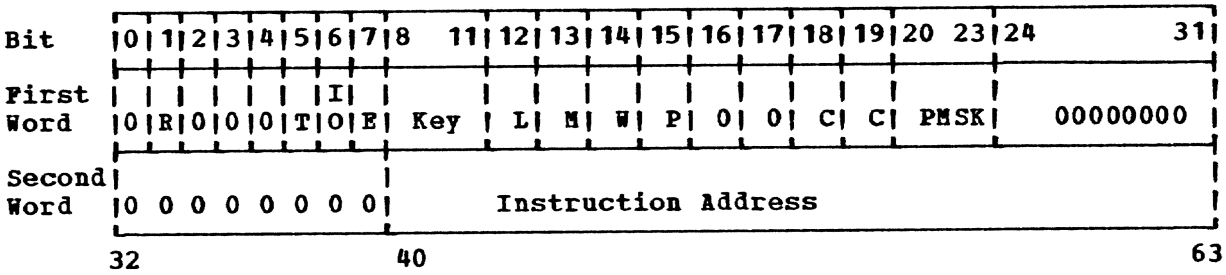
A number of tables, or system control blocks, are also initialized by Startup. These tables enable the resident supervisor to keep track of resources. One of these resources is main storage space, which will be reserved for the resident supervisor's use. Space not used for resident programs, or set aside for their use, is available for allocation to nonresident programs.

NORMAL OPERATION

Extended Control PSW

When Startup transfers control to the resident supervisor, the IBM central processing unit is in the extended control mode, the mode that supports TSS on the System/370 . The format of the extended control program status word (XPSW) is shown in Figure 1. Because resident programs operate unrelocated, bit 5 in the XPSW, the relocation bit, is always 0. The protection key is also 0, giving resident programs access

to all main storage. The privilege state bit is 0, too, since resident programs operate in the supervisor state. Any program interruption in the supervisor state is considered an error; to allow detection of program interruptions, the four program mask bits are 1s. The second word of the XPSW contains the instruction address in bit positions 40-63. Resident programs are responsible for controlling dynamic relocation of non-resident programs; they do not, themselves, run with the address translator on. Addresses used by resident programs are always real addresses, limited by physical storage. The maximum allowable amount of main storage is 16,777,216 bytes (2^{24}).



R	Program Event Recording, disallowed (0) or allowed (1), subject to Event Mask Bits in Control Register 9
T	Relocation, off (0) or on (1)
IO	I/O interruption mask, disallowed (0) or allowed (1) subject to Channel Mask Bits in Control Register 2
E	External interruption mask, disallowed (0) or allowed (1)
Key	Storage protection key
L	Extended Control Mode, BC Mode (0) or EC Mode (1)
M	Machine check mask, disallowed (0) or allowed (1)
W	Wait state, running (0) or waiting (1)
P	Problem state, supervisor (0) or problem (1)
CC	Condition code
PMSK	Program mask
Not used	Bit positions 0, 2-4, 16-17, 24-31, and 32-39; a specification exception is recognized when these bit positions do not contain zeros

Figure 1. Extended control program status word

The Prefixed Storage Area

When the multiprocessing feature is installed in a CPU most addresses associated with storage reference by the CPU are processed by a mechanism called "prefixing." All addresses subject to this processing are referred to as "real" addresses. Storage addresses which are not subject to this processing, and all addresses that have been processed, whether or not they are changed, are referred to as "absolute" addresses.

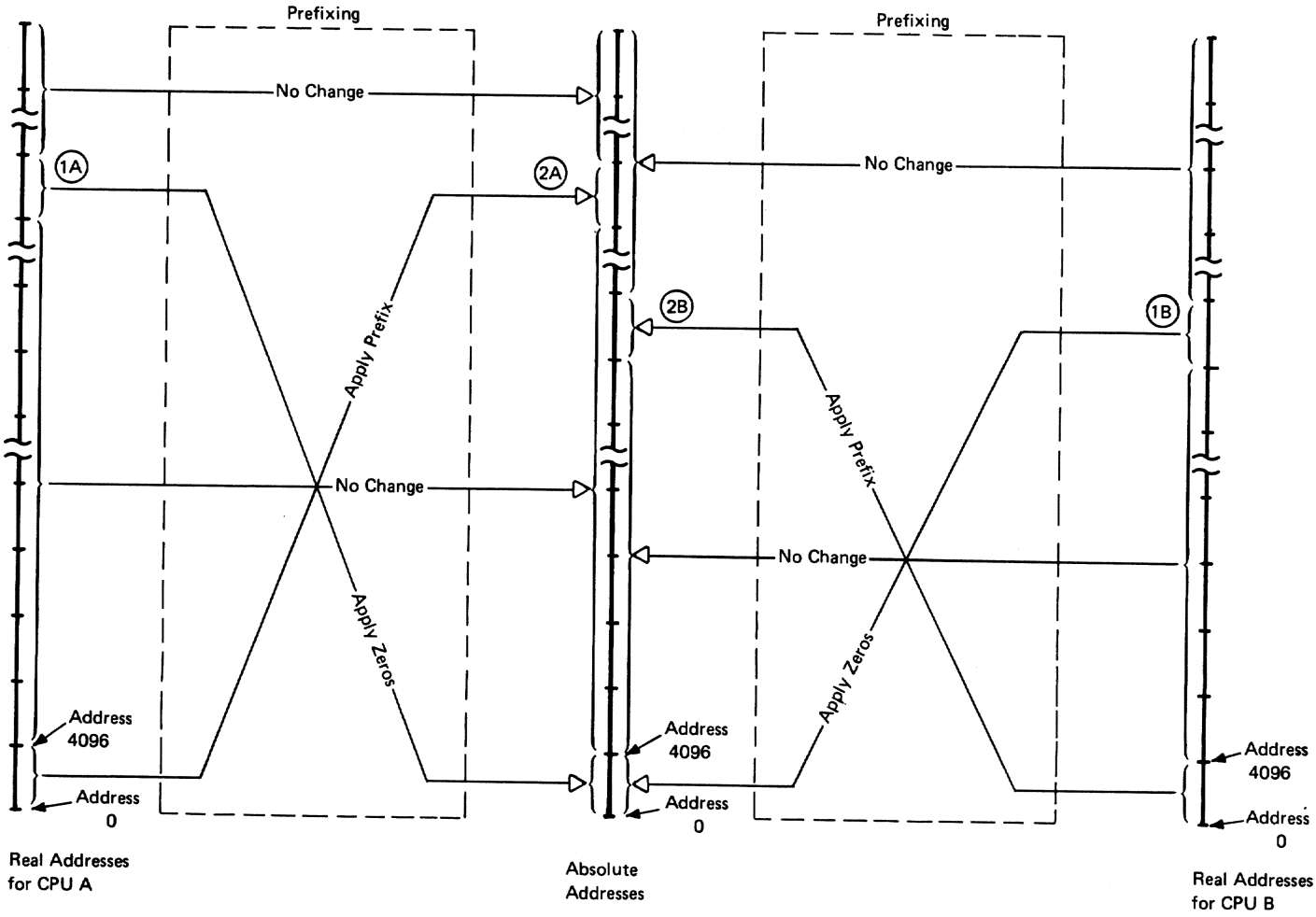
As a result of the processing to form the absolute address, real addresses 0-4095 are interchanged with the 4,096 addresses of the block that begins at the address identified in the prefix register. All other real addresses remain unchanged.

The real addresses 0-4095 include the addresses of the assigned storage locations that are implicitly generated by the CPU and channels, and includes the addresses that can be specified by a program without the use of a base address or an index. Prefixing provides the ability to reassign this block of real locations for each CPU to a different block in absolute main storage, thus permitting more than one CPU shar-

ring main storage to operate concurrently with a minimum of interference, especially in this processing of interruptions.

Because the prefixing mechanism interchanges the real addresses, each CPU can access all of absolute main storage, including the first 4,096 bytes and the assigned locations for another CPU.

The relationship between real and absolute addresses is graphically depicted in Figure 2.

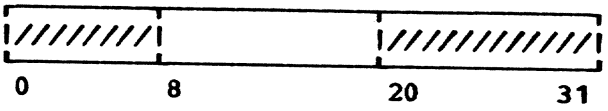


(1) Real addresses in which the high-order 12 bits are equal to the prefix for this CPU (A or B).

(2) Absolute addresses of the block that contains, for this CPU (A or B), the assigned locations in real storage.

Figure 2. Relationship between real and absolute addresses

The prefix is a 12-bit quantity located in the prefix register. The register has the following format:



The contents of the register can be set and inspected by the privileged instructions SET PREFIX and STORE PREFIX, respectively. On setting, bits corresponding to bit positions 0-7 and 20-31 of the prefix register are ignored. On storing, zeros are provided for these bit positions. The prefix register is initialized to zero.

Prefixing is applied to all references to main storage and to keys in storage, except for references by a CPU to the permanently assigned storage locations during performance of the store-status function, and except for references by a channel to extended-logout locations, to I/O data, to indirect-data-address words, and to CCWs. When dynamic address translation is specified, prefixing is applied after the address has been translated by the dynamic-address-translation mechanism. When installed, prefixing is always active and is not subject to any mode control.

When prefixing is applied, the storage address is translated as follows:

a. Bits 8-19 of the storage address, if all zeros, are replaced with bits 8-19 of the prefix.

b. Bits 8-19 of the storage address, if equal to bits 8-19 of the prefix, are replaced with all zeros.

c. Bits 8-19 of the storage address, if not all zeros and not equal to bits 8-19 of the prefix, remain unchanged.

In all cases, bits 20-31 of the storage address remain unchanged.

Only the address presented to storage is translated by prefixing. The contents of the source of the address remain unchanged.

The distinction between real and absolute addresses is made even when prefixing is not installed or when the prefix register contains all zeros. In both of these cases, a real address and its corresponding absolute address are identical.

The format of the prefixed storage area can be seen by copying the dsect CHAPSA.

SUMMARY

Resident programs make up the part of TSS known as the resident supervisor. The extended control mode of operation is normal for resident programs. These programs, operating in the supervisor state with the address translator turned off and with an XPSW protection key of zero, can execute any System/370 (including all M158 and M168 multiprocessors) instructions and use all main storage except page zero in a dual system. Each processing unit also has 16 general-purpose registers, 4 floating-point registers, 16 extended-control registers, an interval timer, an address translator, and other components for fetching and executing instructions.

NONRESIDENT PROGRAMS

Nonresident, or virtual storage, programs are programs that operate with the address translation unit turned on; they do not permanently reside in main storage. There are two kinds of virtual storage programs: privileged and nonprivileged. For a conceptual understanding of virtual storage, see Concepts and Facilities and System Logic Summary.

TASK STRUCTURE

The nonresident portion of TSS, as well as a user's application programs, operate within the context of individual tasks. To the system user, the task is an individual work requirement; to the system itself (and the resident supervisor in particular), the task is a unit of activity to be allocated system facilities, including a periodic time-slice of the CPU.

A task has a virtual storage whose size is essentially independent of the physical main storage available to the resident supervisor. Virtual storage is organized into pages of 4096 bytes, which are further collected into segments of 16 pages. Virtual storage consists of a maximum of 256 segments (16,777,216 bytes) with 24-bit addressing.

Just as OS subdivides its instruction set into two states, supervisor and problem, a task may run with a privileged or a nonprivileged instruction set. The privileged instruction set, somewhat analogous to the supervisor state, contains all OS problem state instructions and those SVC (Supervisor Call) instructions designated as privileged. The nonprivileged instruction set, somewhat analogous to the problem state, contains all OS problem state instructions and those SVC instructions designated as nonprivileged.

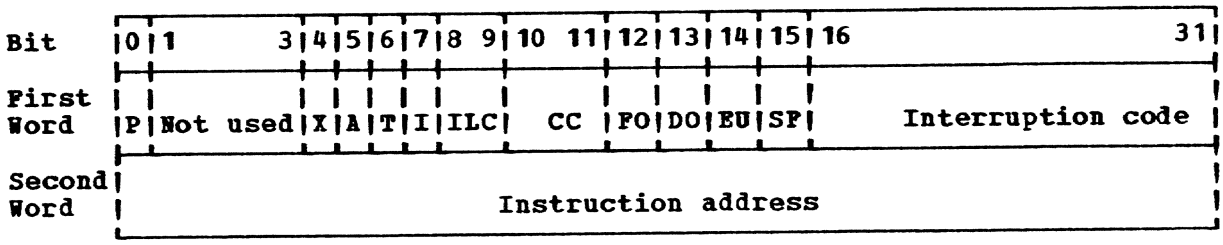
Each task contains not only any application programs called by the user (in the form of object modules), parts of data sets referred to by these programs, and dynamically acquired virtual storage data areas, but also those portions of the TSS control program which are nonresident and any system facilities specifically requested (such as language processors). But each task does not have to have a separate copy of the nonresident system modules (the nonresident TSS control program); it shares with other tasks the a single copy of nonresident system modules in main storage.

INITIAL VIRTUAL STORAGE

The task monitor and a number of programs (both privileged and nonprivileged) are collected into what is called initial virtual storage or initial virtual memory (IVM). Startup establishes initial virtual storage by constructing a standard set of segment and page tables to be used by each newly created task. Initial virtual storage programs are never dynamically loaded; they are permanently resident in virtual storage. Of course, IVM programs are paged in and out of main storage. All other privileged programs are brought into virtual storage, as required, by the dynamic loader (which must be part of IVM). Virtual storage is never "empty"; it always contains at least the programs that make up the IVM.

Virtual Program Status Word

Each task has 16 general-purpose registers and 4 floating-point registers available to it. The status of a task is described by its virtual program status word (VPSW), which is shown in Figure 3.



P Privilege. (0) privileged or (1) nonprivileged

Bits 4-7 are the task mask and are interpreted:

X External interruptions
A Asynchronous interruptions
T Timer interruptions
I Synchronous interruptions

ILC Instruction length code
CC Condition code

Bits 12-15 are interpreted:

FO Fixed point overflow mask
DO Decimal overflow mask
EU Exponential overflow mask
SF Loss of significance mask

For all of the above masks, one permits an interruption on the occurrence of the condition and zero inhibits the interruption.

Figure 3. Virtual program status word

Interruption Storage Area

A task is interrupted by a virtual, or task, interruption. When this occurs, the information constituting the current VPSW is stored in a predetermined area of virtual storage (depending on the type of interruption), becoming the old VPSW. A new VPSW, obtained from another location in that area, becomes the current VPSW. The area, called the interruption storage area (ISA) is analogous to the prefixed storage area (PSA) in the system's real main storage. The interruption storage area is bytes 0 through 8190 of virtual storage.

There are eight different virtual, or task, interruptions: program, supervisor call, external, asynchronous I/O, task-timer, synchronous I/O recoverable data set paging error, and VSS. The occurrence of four of these interruptions is controlled by the task mask in the VPSW, analogous to the system mask in the PSW. If the mask bit corresponding to a given interruption type is 0, or if the interruption storage area is locked (ISALCK set by the ITI macro instruction), interruptions for that type are saved by the resident supervisor, until the mask bit is set to 1.

Each task has an instruction set consisting of all OS problem state instructions and a number of supervisor call instructions. The supervisor call instructions are further divided into SVCs that can be issued only by privileged programs, SVCs that can be issued only by privileged or nonprivileged programs depending on the authority code of the programmer. The privileged SVCs are analogous to OS supervisor state instructions. Each of these SVCs is described in detail later.

The current VPSW, contained in field ISACVP, includes the address of the instruction following the last instruction executed prior to the in-

terruption. While the interruption is being serviced or is waiting to be serviced, this address is significant in that it points to the instruction at which execution is to be resumed. Once execution is resumed, the current VPSW continues to point to the same instruction; the address is not incremented as each instruction is executed and, therefore, loses its significance.

STORAGE PROTECTION

Although the virtual program status word doesn't contain a key, storage protection is in effect for virtual programs. The resident supervisor assigns storage keys to virtual programs when it creates external page table entries for them; it sets keys in the main storage pages it allocates (see the ADDPG and ADSPG macro instruction descriptions). Each main storage page is assigned a storage protection key; Figure 4 illustrates these assignments.

Type of Page	Key	Fetch Protection Bit
Nonprivileged read/write	1	off
Nonprivileged read-only	2	off
Privileged	2	on
Engaged in paging operation	3	off
Storage obtained from supervisor core allocation	4	off
Resident supervisor	5	on

Figure 4. Main storage page key assignments

The ability of a CPU or a data channel to have access to main storage is controlled by the protection key contained in storage and the privilege key used by the CPU (PSW) or data channel (CAW). The resident supervisor assigns keys to programs and channel programs before starting them; these assignments are shown in Figure 5.

Category	Key
Processing Unit Programs	
Nonprivileged	1
Privileged	0
Resident supervisor	0
Data Channel Programs	
Nonprivileged I/O	1
Privileged I/O	2
Paging I/O	3
IORCB I/O	4
Sense data I/O	4

Figure 5. CPU and data channel key assignments

Figure 6 shows the significance of various combinations of PSW and storage keys and the programs to which they may be assigned. Those within the heavy line designate nonprivileged user key combinations. The other combinations are available only to privileged system programs. Storage protect key 2F is the same as key 2 but with fetch protection added.

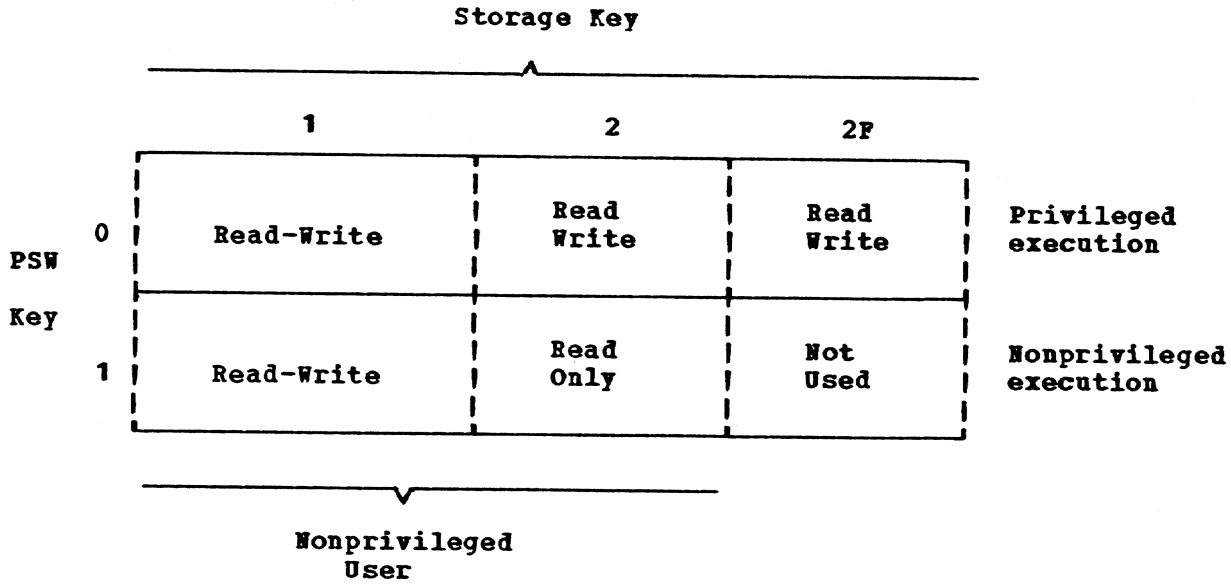


Figure 6. PSW and storage protection keys

PRIVILEGED PROGRAMS

A privileged program is a virtual program recognized by having its virtual program status word (VPSW) privilege bit (bit 0) set to 0, analogous to the privileged state bit (bit 15) in the real PSW. A privileged program differs from a nonprivileged user program in two principal ways: it operates with a PSW protection key of 0 and it may issue most Supervisor Call instructions. A privileged program can access all virtual storage in its own task; it cannot access private virtual storage in other tasks.

Privileged programs exist to provide services to nonprivileged programs. Privileged service routines that can be called by users are "connected" to the task monitor through a table, called the ENTER table (see the ENTER macro instruction); other privileged service routines are closed subroutines used only by privileged callers.

NONPRIVILEGED PROGRAMS

A nonprivileged program operates in a task. The storage of this task contains all programs that make up IVM and any other programs that have been brought into virtual storage by the dynamic loader. A nonprivileged system program may be part of IVM (the assembler is an example of a nonprivileged initial virtual storage program).

A nonprivileged program may use any OS problem state instruction and any nonprivileged Supervisor Call instruction. Nonprivileged system programs may not, in general, use the privileged Supervisor Call instructions. The resident supervisor uses the task status index to determine whether a program can issue privileged SVCs.

Since a number of SVC codes are used by the resident supervisor, a kind of substitute SVC, called ENTER, is used for most transfers of control from nonprivileged to privileged programs. A nonprivileged program can't transfer control to a privileged program with a branch instruction since all privileged programs are fetch protected from all nonprivileged programs (both system and user). ENTER codes, analogous to SVC codes, are used by the task monitor to determine where to transfer control.

The programming conventions that system programmers must observe when coding modules that are to be made a part of the resident or nonresident portion of TSS control programs are discussed below.

RESIDENT (SUPERVISOR) PROGRAMS

CONVENTIONS FOR NAMING OBJECT MODULES

All TSS programs have standardized object module names, control section names, and entry point names. When an object module becomes part of the system, any reference to it must use the module name, an entry point name, or a control section name. TSS object module names consist of five characters; those TSS modules that are part of the resident supervisor have names with the form:

```
CEAxx
```

where xx are alphameric characters that identify the module within the resident supervisor. All TSS object module names begin with C; the characters EA identify resident supervisor modules.

All entry point and control section names begin with the module name, like this:

```
CEAxxn
```

where n is a character that identifies the entry point or control section within the object module. Note that special characters are not used in TSS names.

As an example, the pathfinder module in the resident supervisor has the name CEAA5; its entry points are CEAA5P, CEAA5R, and CEAA5S.

There are other sets of modules that are in the resident part of TSS. These components and their naming conventions are:

Resident RTAM	CEDxx
Resident RSS	CEHxx
Machine check recovery	CMAxx

MODULE DESIGN CONSIDERATIONS

Assume that you have a change to TSS in mind, and that you clearly understand the logic of the change you wish to make. How do you construct the program? Resident programs are different from nonresident programs in one major way: Resident programs do not contain prototype control sections (PSECTs). The purpose of a prototype control section is to contain any part of a program that changes during relocation or execution. As pointed out previously, resident programs are never relocated or never change addresses during execution. The address constants used by resident programs are resolved during startup; they will not be changed after the system is initialized. The only items in the resident supervisor that can change are the variables used by resident programs. These variables are kept in registers, system control blocks, or working storage obtained from the supervisor core (main storage) allocation subroutine. Your program must be designed to use one of these areas for holding variable information; which one you use depends on what you are

attempting to do. The key test of a resident program's correct construction is that it be simultaneously executable by multiple processing units. If the registers are used as working storage, multiple processors may simultaneously execute the program, since each processor supplies its own registers. If a system control block is used, the lock byte controls the modification of variable information. If storage obtained from the supervisor core allocation subroutine is used, each allocation of storage is kept separate from the others to ensure the protection of variables.

Module Structure

Like any other object module, a resident module consists of a program module dictionary (PMD) and text. Usually, a resident program contains a single read-only, nonprototype control section. A resident program may contain address constants to be computed and placed into the text by Startup. The read-only control sections do not change; they are never modified during program execution.

In addition to read-only control sections, the resident supervisor contains tables or system control blocks. A system control block is data in main storage, organized in some way known to the programs that use it. The system table, CHBSYS, is an example of a system control block used by a number of resident programs; it contains such information as parameters used for task migration, limits of task size and number, and other values affecting the overall operation of the system.

If there is any possibility that one processor can change a system control block at the same time another processor is working on it, the control block must be protected, or interlocked, with a lock byte. A lock byte is a single byte used to control access to variable information. The test and set instruction is used to find out whether a lock byte is on or off (and also to turn it on). For example:

```
TEST    TS    LOCK
        BO    WAIT
```

tests a lock byte called LOCK; if LOCK is all 1's, control is transferred to WAIT. A lock byte is set (or on) if it is all 1's (actually, only the high-order bit is tested); it is reset (or off) if it is all 0's.

Since the correct locking and unlocking of TSS lock bytes is critical to the running of a multi-cpu system, TSS programs should use two macros, GETLOCK and FREELOCK, to perform these functions. These macros generate code, provide some tracing data for debugging and generate system errors in a standard way.

Some programs do not need to test lock bytes, since they are subroutines of programs that do test. Some control blocks are not individually tested (and do not contain a lock byte) but are gathered into queues; the entire queue is interlocked instead of its members.

A lock byte is reset when the program that set it has finished using the protected information. In most cases a second processing unit will only wait a certain length of time for a lock byte to be reset; if the lock byte is not reset within that time period, a minor system error is recognized. When control is returned to the point of interruption by the error routine, the protected data becomes available.

In addition to read-only control sections and interlocked system control blocks, the resident supervisor contains a number of pages (or a "pool") of main storage that may be used, as required, by resident programs. The program controlling the use of this storage pool is the supervisor core allocation module. Since the supervisor core allocation

module itself cannot require the allocation of storage space to free working registers, it saves the registers in a special area in the PSA.

Relatively few system control blocks continuously require main storage space; most have transient storage needs. Resident programs usually obtain storage space for transient data from the supervisor core allocation module. When the need for this data no longer exists, the space is returned to the supervisor core release subroutine; this dynamic allocation of main storage space ensures that the resident supervisor doesn't tie up more storage space than it actually needs. Most transient data areas cannot be used simultaneously by separate processing units; these control blocks are not interlocked. A few transient data areas, however, can be used by separate processing units; these are interlocked.

Some data areas are known only to one processing unit because one of its registers points to the data area; other data areas are known to all processing units because the address of the data area is kept in common main storage.

Getting Resident Working Space

Working space is not assembled into resident programs for two reasons: (1) it is inefficient to assemble space that may not be used into a program, and (2) the resident program would have to schedule the use of that space if the program were to be simultaneously executed by separate processing units.

Resident programs use four modules to obtain working space for their execution. These routines are supervisor core (main storage) allocation (CEAL01), supervisor core release (CEAL02), user core allocation (CEANB), and user core release (CEAL04). User main storage is allocated from one pool and supervisor main storage from another, but the supervisor pool may be replenished from the storage released by user core release. The supervisor core allocation routine satisfies requests for resident working space such as control blocks like the generalized queue entry (GQE) and the task status index (TSI). The user core allocation routine satisfies requests for storage for extended task status indexes (XTSI) and for nonresident program pages.

The supervisor core allocation subroutine is a special program, since it has private space in the prefixed storage area (PSA), which it uses to store the contents of the general registers. It must use this area since there is no subroutine that it can call to get working space. Programs that call the supervisor core allocation subroutine must save registers 0, 1, 14, and 15 before transferring control. They can't do this without working space, though, so four words in the prefixed area (PSASCU) are set aside for programs calling the supervisor core allocation subroutine. This lets a called program immediately become a calling program without losing the contents of any of the general registers that were supplied to it. Since a program calling supervisor core allocation must use registers 0, 1, 14, and 15 to transfer control (and parameters), it would lose the original contents of these registers if it had no place to save them. A typical use of the supervisor core allocation subroutine might look like this:

```
SUBR    USING   *,15            REGISTER 15 CONTAINS BASE
        COPY    CHAPSA          GET THE DSECT
        USING   CHAPSA,0        PSA DSECT NEEDS NO BASE
        CSECT                   REESTABLISH CSECT
        STM     14,1,PSASCU     SAVE REGS 0, 1, 14, and 15 IN PSA
        LA      0,128           REQUEST 128 BYTES
        SR      1,1             OPTIONS ALL ZERO
        L       15,ADCON        POINTER TO SUPVR CORE ALLOC
        BASR    14,15           TRANSFER CONTROL
RTRN    LR      8,1             SAVE ADDRESS
        LM      14,1,PSASCU     RESTORE REGS 14, 15, 0, AND 1
        •
        •
        •
ADCON   DC      V(CEAL01)       ADDR SUPVR CORE ALLOCATION ROUTINE
```

In this example, when supervisor core allocation returns control, register 1 points to a 128-byte area of main storage that can be used for any further transient storage needs this program may have. The supervisor core allocation subroutine will not disturb the register contents saved by this program in PSASCU since supervisor core allocation has its own save area in the PSA (PSACAS).

To give this space back to the supervisor core release module, the program might be coded like this:

```
DONE    STM     14,1,PSASCU     SAVE REGS
        LR      1,8             ADDRESS OF SPACE WE'RE RETURNING
        LA      0,128           GIVE BACK 128 BYTES
        L       15,ADCN2        POINTER TO SUPVR CORE RETRN
        BASR    14,15           TRANSFER CONTROL
RTRN    LM      14,1,PSASCU     RESTORE REGS
        BR      14              RETURN TO ORIGINAL CALLER
ADCN2   DC      V(CEAL02)       ADDR SUPVR CORE RELEASE
```

This will return for reallocation the 128 bytes obtained in the previous example.

Two macros, GETCORE and RELCORE, are used in TSS to standardize and simplify these linkages.

For transient work areas that will not be needed after the module returns to the program that called it, e.g., register save areas, message areas, parameters for a subroutine, etc., a macro, GETWORK, is to be used. This macro allocates space from a preallocated stack. It uses far less instructions than calling supervisor core allocation and the space does not have to be released.

Secondary Entry Points

Resident modules with more than one entry point are designed so that their base register always points to the primary entry point, even if control of the modules has been transferred to a secondary entry point. Sometimes it's done like this:

```
BASE      EQU       15
ENTRY1    USING     *,BASE
          .
          .
          .
ENTRY2    BASR      BASE,0
          USING     *,BASE
          L         BASE,ADCON
          USING     ENTRY1,BASE
          .
          .
          .
ADCON     DC        A(ENTRY1)
```

System Control Blocks

The resident supervisor consists of three parts: read-only control sections, system control blocks, and a pool of dynamically allocatable main storage. The resident supervisor uses the storage pool to create transient system control blocks such as the generalized queue entry (GQE) and the page control block (PCB). During their "lifetimes," transient control blocks are resident in main storage. Transient control blocks exist only as long as they are needed; this may be a few milliseconds or a few minutes. When they are no longer needed, the main storage space they occupy is returned for reallocation; they are not paged out to auxiliary storage.

The resident supervisor also creates nonresident system control blocks. Nonresident system control blocks exist on auxiliary storage and are brought into main storage only when needed. They exist on some storage device for a relatively long time, for example, for an entire terminal session. Their time in main storage may represent only a small fraction of their lifetime in the system.

System Control Block Names

A system control block usually requires two names: the name of the dummy section (DSECT) that describes its format, and the symbolic address that points to the information described by the dummy section. All TSS programs, resident and nonresident, use the same rules to name system control blocks. A dummy section name looks like this:

```
CHAxxx
```

The characters xxx identify the dummy section. All fields used within the dummy section look like this:

```
xxxfff
```

The characters xxx are the same as the last three characters of the dummy section name. The characters fff are any three characters that identify the field within the dummy section. For example, these are the assembler statements for a typical dummy section:

```
CHAABC    DSECT             CONTROL BLOCK NAME
ABCFCA    DS      1F        FIELD NAME
ABCRJG    DS      4F        FIELD NAME
ABCXYZ    DS      3C        FIELD NAME
ABCFLG    EQU     ABCXYZ    FLAG NAME
ABCFLGM   EQU     X'80'     FLAG MASK
```

Note that the field name, ABCFLG, is the name of a byte containing a flag bit. The field ABCFLGM can be used as a mask byte in a test under

mask instruction (TM) to test the condition of the flag. Mask names are of the form:

```
xxxfffM
```

where xxxfff is the name of the dummy section field to which the mask is applied. For more information on using a dummy section, see "Dummy Sections" below.

The dummy section is, of course, only a description of information; it does not supply anything more than the format of the information it describes. Symbolic addresses that point to non-transient areas of storage described by dummy sections are named like this:

```
CHBxxx
```

where the characters xxx are the same as the last three characters of the dummy section name. For example:

```
DATA      DC    V(CHBABC)
```

DATA contains an address constant pointing to an area of storage organized as described by the dummy section CHAABC.

Remember, CHAxxx says what the information looks like; CHBxxx says where the information is located. The DSECT can be used for both non-transient system control blocks and for transient areas (which cannot use CHBxxx). Symbols generated by macro instructions always begin with CHD. For example:

```
CHD103    MNOTE 3,'ERROR'
```

might be found in a macro definition.

Dummy Sections

The dummy section (DSECT) is used extensively throughout TSS so that parts of the system can refer to commonly used data items by symbolic names. You can refer to a field in a system control block by the name assigned to that field in the dummy section; this frees you from having to use the field's numeric location. (Actually, the dummy section supplies a number of symbolic field names, lengths, and relative positions which the assembler translates into numeric displacements.) You needn't worry about the specific physical structure of the system control block to which you are referring if you use a DSECT to describe the control block. All you need be concerned with is the field structure (bit, byte, halfword, etc.). You don't care where the field is located within the system control block. Thus, if the field position changes, but the field length, boundary alignment, and the meaning of its contents don't change, your program will still run properly after it is reassembled. Reassembly is necessary since displacement values may have changed as a result of using the new dummy section.

In TSS, the dummy section is more than a programmer convenience. Dummy sections for system control blocks, obtained from the assembler copy/macro library, ensure that all programs using the same system control block use the identical format. The set of TSS dummy sections can be viewed as a central, current description of all system control blocks.

A typical TSS dummy section is illustrated in Figure 8 under "Linkage Convention," later in this section. Several conventions apply to dummy sections. These conventions minimize the need for redesigning programs if the dummy sections they use are changed. Dummy section fields that are integral multiples of bytes in length are referred to in a program

by using the name of the field. Fields that are less than one byte long are referred to by using a mask; we must do this because the central processing unit cannot directly address a field shorter than a byte. The name of the mask associated with a field that is less than one byte long is obtained by adding the character M to the field name. If we wanted to determine whether the field VPSAI were a 1, we might write:

```
TM    VPSAI,VPSAIM      TEST UNDER MASK
BZ    FIELDOFF          BRANCH IF ZERO
```

The programmer doesn't have to know where the field VPSAI is located within the system control block or what bit pattern defines the mask VPSAIM. This information is supplied by the dummy section which he incorporates into his program from the assembler copy/macro library. The field he is testing with the Test Under Mask (TM) instruction need not be restricted to a single bit. It can be any combination of up to eight bits, as long as all the bits fall within one byte. The conditional branch instruction can be used to determine if the bits he is testing are all 1's, all 0's, or mixed.

For bit fields, the field name is always the name of the byte in which the bits appear. Since a single byte can have up to 256 different conbinations of bits, a single byte could have up to 256 different bit fields. We frequently find, therefore, that the names of different bit fields are synonomous; that is, they point to the same byte. The bit mask corresponding to the field name must be used to extract the proper bits.

The dummy section itself does not take up program storage space; it is used exclusively to describe a storage area to which it is applied. To properly use a dummy section, first load a register with an address constant pointing to a storage area containing information described by the dummy section. Then issue a USING statement to tell the assembler that the corresponding dummy section format is to be applied to the storage area pointed to by the register given in the USING statement. It looks like this:

```
L       5,ADCON
USING   CHAVPS,5
```

assuming that ADCON has been defined as:

```
ADCON DC V(WORKAREA)
```

This would apply the format given by CHAVPS to the storage area beginning at the symbolic location WORKAREA.

You can define your own dummy sections and use them as you see fit. In most cases, though, you will get the dummy section from the assembler copy/macro library (see Section 4, "Generating and Maintaining TSS") by issuing a COPY statement with the name of the dummy section as the operand. Here is an example:

```
COPY    CHAVPS
```

The dummy section on the assembler copy macro-library is included in your program at the point of the COPY statement. This enables you to symbolically refer to the system control block CHAVPS (see System Control Blocks).

ENABLING AND DISABLING INTERRUPTIONS

Because they operate in the supervisor state, resident programs can enable and disable interruptions by setting and resetting the system

mask or by altering the contents of extended-control registers. The instruction

```
SSM  =X'00'
```

sets bits 0 through 7 of the extended program status word to zero. The processing unit affected interprets these bits as: program event recording off, address translator off, and I/O and external interruptions disabled. To restore interruptions,

```
SSM  =X'03'
```

is interpreted by the processor as: program event recording off, address translator off, and I/O and external interruptions enabled. If you wish to modify the extended-control registers, the instructions:

```
STCTL 2,2,SAVE
L     6,SAVE
N     6,=X'BFFFFFFF'
ST    6,SCRATCH
LCTL  2,2,SCRATCH
```

save the contents of control register 2 and disable interruptions from channel 1 (as viewed by the processing unit issuing the LCTL). The instruction:

```
LCTL  2,2,SAVE
```

restores the original contents of control register 2. The work areas for SAVE and SCRATCH would be obtained using the supervisor core allocation subroutine.

SUPERVISOR LINKAGE CONVENTIONS

A resident program links to subroutines by using (1) a V-type address constant or (2) an A-type address constant and an EXTRN statement. Startup resolves all symbolic references among resident programs, and supplies the correct values for the address constants. Resident programs never use R-type address constants (they do not contain prototype control sections). One resident program transfers control to another by a Branch-and-Store (BASR) instruction. Any understanding between the calling and the called programs concerning the contents of the general registers is arbitrary and depends on the particular programs involved. The calling program must know what register the called program is using as an entry base register. To transfer control, the calling program loads the address of the called program's entry point into the called program's entry base register, like this:

```
L         BASE,=V(entry point name)
```

Then the calling program branches to that entry point:

```
BASR      14,BASE
```

The called program expects its entry base register to contain the address of its entry point. The called program usually begins like this:

```
USING     *,BASE
```

to tell the assembler that register BASE contains the address of the called program's entry point when control is transferred.

To standardize program linkage, TSS resident supervisor uses a Communication Vector Table (CVT). The CVT contains all the entry points for TSS supervisor modules. A macro, RCALL, will load the entry point address in register 15 and link to the called program with a BASR 14,15.

To standardize linkages and provide some traceback information, two macros, SAVER and RETRNR, are used. These are used in the same area as that used by the GETWORK macro.

Programming Convention Comments

There is no requirement for resident programs to use particular registers as base registers, return registers, or parameter registers; however, almost all resident programs use these registers:

Register 0 -- parameter register
Register 1 -- parameter register or address of parameter list
Register 14 -- return address of calling program
Register 15 -- entry point of program being called

Because of these register assignments, most programs being called begin with:

```
         USING     *,15
```

and end with:

```
         BR        14
```

or the equivalent.

A number of resident programs, such as SVC processing routines, return control to a location pointed to by a V-type address constant instead of branching to the address contained in register 14. For example:

```
THRU     L         14,ADCN3      GET RETURN ADDRESS
         BR        14            TRANSFER CONTROL
ADCN3    DC        V(CEAHND)     ADDR SVC Q PROC RETURN
```

is the way that an SVC processing routine can transfer control back to the supervisor call queue processor. This is done because most SVC processors require two common functions to be performed, and this portion of the SVC queue processor provides them with the functions.

NONRESIDENT PROGRAMS

PRIVILEGED PROGRAM CONVENTIONS

Naming Conventions

As discussed in the section about naming resident supervisor programs, all TSS module names begin with the letter C. All privileged module names have the form:

CZxxx

where the characters xxx identify all module names beginning with CZ. All control section names and entry point names of nonresident privileged modules add a character to the end of the module name to form a unique entry point or control section name. This is analogous to the way entry point and control section names are formed for resident supervisor modules. For example, an entry point of privileged module CZCJT might be CZCJTH.

Dummy sections for system control blocks are used by nonresident privileged programs in the same way that they are used by resident supervi-

sor programs. All system dummy section names begin with CHA; the location of the first byte of data described by a system dummy section is named by a label beginning with CHB. CHAXYZ is a dummy section describing data located at a virtual storage address equivalent to CHBXYZ.

The dynamic loader treats all external symbols beginning with the characters SYS as system names. (A control section without either the PRVLGD or SYSTEM attribute cannot define system names as external symbols.) (See Figure 41 for the effect of authority codes in dynamic loader processing.)

Writing Privileged System Programs

Virtual storage (that is, nonresident) system programs are divided into two classes: programs that make up initial virtual storage and programs that are dynamically loaded. Initial virtual storage (IVM) is composed of all those system programs (both privileged and nonprivileged) needed to dynamically load a program and those system routines that are frequently used by an installation. Privileged programs that are not part of IVM are brought into virtual storage, as required, by the dynamic loader and the miscellanous programs it uses for assistance.

In writing a system program, you must know whether a program to be called is in IVM or not. Any program in IVM should not be explicitly called since this causes unnecessary system processing.

The use of the E option in the CALL requires action by the dynamic loader; only programs outside of IVM may use CALL with the E option.

Almost all TSS programs can be shared by several users. When a program is shareable, or public, it must be put together in a special way. Each public program is thought of as consisting of two parts.

One part is made up of all the instructions and data in the program that never change because of relocation in virtual storage by the dynamic loader or because of execution by a processing unit (variables). This part of a public program is constant; it never changes under any circumstances.

The second part of a public program consists of those parts that may change because of relocation or execution: the program's address constants and variables.

The parts of a public program that may change -- the address constants and variables -- are collected in a prototype control section (PSECT). All other control sections of a public program should be given the attribute READONLY, since they can never be modified. Exceptions are the tables, such as the symbolic device allocation table (SDAT), that are protected with lock bytes and are shared, nonread-only control sections. The division of a public program into prototype control sections and read-only control sections allows a number of different tasks to share the same program without destroying one another's results. This is accomplished by giving each task that is sharing the public program its own private copy of the prototype control section, while allowing each task to share a single copy of the read-only control sections. In this way, each task has a private copy of those parts of the public program that may change, thus preventing tasks from destroying one another's variables and allowing each task to have its own address constants.

You should take care not to confuse intertask reenterability with intratask reenterability. The use of prototype and read-only control sections permits programs to be shared among many different tasks; this is intertask reenterability. The use of a prototype control section for storing variables does not automatically guarantee that, within a single

task, a program can be reentered. All programs are freely interruptable by any real (not virtual) interruption. When such an interruption occurs, before control is returned to the interrupted program in virtual storage, the resident supervisor checks whether there are any pending task-interruptions. If there are pending task-interruptions, if the corresponding task-mask bit in the virtual program status word is set to 1 (enabling task-interruptions) and if the ISA lock byte is zero, control is returned not to the interrupted program, but to the task monitor. The task monitor, after some housekeeping, transfers control to the appropriate task-interruption-handling routine. In some instances, the interruption handler may have to use the interrupted program as a subroutine. When this happens, the interrupted program is being reentered. It is thus task-interruption sensitive and it must be constructed to allow for this sensitivity. The prototype control section is no help in permitting intratask program reenterability since, within this single task, there is only one prototype control section for each public program and only one copy of variables and address constants can be preserved in it.

Although address constants change as a result of program relocation and are placed in a public program's prototype control section, and may assume different values from task to task, they are not considered variables within a task. Once supplied by the dynamic loader (or by startup for IVM), an address constant within a given prototype control section will not change.

Within a single task, we are concerned about those parts of a program (public or otherwise) that change as a result of that program's execution by a processing unit. If a program that stores variables in fixed areas of virtual storage can be called by a number of other programs, it must protect itself against task-interruptions. If a program must be interruptable (by task-interruptions), it must use GETMAIN (or something equivalent) to dynamically allocate virtual storage and thus prevent the accidental destruction of variables. GET and PUT are examples of programs that can be in use by one program, interrupted, and reentered for use by another program within the same task.

If, in a privileged program, you want to disable task-interruptions during some processing, you can use the macro instruction ITI (inhibit task interruptions); to enable task-interruptions, the macro instruction PTI (permit task interrupts) may be used (see Appendix A). For example:

```
LOCK    ITI             DISABLE TASK INTERRUPTIONS
          .             MISCELLANEOUS INTERRUPTION-SENSITIVE CODING
          .
          .
        PTI             ENABLE TASK INTERRUPTIONS
```

shows how task-interruption might be disabled and restored in a program.

Excluding dummy sections, which are not true control sections (see "Dummy Sections"), you may have two kinds of control sections in your program: prototype (PSECT) and nonprototype (CSECT). From the standpoint of the dynamic loader, there is very little difference between a PSECT without qualifying attributes and a CSECT without qualifying attributes. Throughout TSS, however, PSECTs are used in public programs to contain address constants and variables; you should think of prototype control sections as the private part of shared program modules.

Be careful not to confuse the attributes PRIVLGD and SYSTEM. PRIVLGD includes SYSTEM; every privileged program is a system program as far as the dynamic loader is concerned. SYSTEM does not necessarily include PRVLGD, however; every system program is not privileged.

You might code a sample privileged program like this:

```
            TITLE    'SAMPLE PRIVILEGED PROGRAM'
            DCLASS   PRIVILEGED                    THIS ALLOWS PRIV
                                                   MACRO EXPANSIONS
            COPY     CHAISA                        GET FORMAT OF
                                                   ISA
CZABP       PSECT    PRVLGD                        PUT ALL THE
                                                   ADCONS AND
                                                   VARIABLES HERE
            EXTRN    CHBXYZ                        LOCATION OF
                                                   TABLE XYZ'S DATA
CACABC      CSECT    READONLY,PUBLIC,PRVLGD        PURE PROCEDURE
              .                                    SECTION ANYTHING
              .                                    HERE BUT ADCONS
              .                                    AND VARIABLES
              .
            END      CZABC
```

NONPRIVILEGED PROGRAMS

There isn't a great deal of difference between a privileged and a nonprivileged system program; almost everything written above about privileged system programs applies to nonprivileged system programs. The most significant difference between them is that nonprivileged system programs operate with an extended program status word protection key of 1; they cannot read or write privileged control sections.

PROGRAM DESIGN CONSIDERATIONS

In thinking about nonprivileged programs, don't confuse the privilege of a program with the authority of the programmer who directed that the program be loaded. Despite any assembler language declarations, any program you write is implicitly a system program as long as you log on using your user ID with a P or 0 authority code. Remember that all sections you load using your P or 0 authority code are private; the dynamic loader ignores the PUBLIC attribute. Only a task having a U authority code can load a control section having the PUBLIC attribute, and then only if the module containing the control section is in a shared data set. (A module requiring more than 256 shared pages will not be loaded as public, however; the module will be loaded with private pages.)

A fence-sitter is a system module that assumes the privilege of the calling module. Such a module should not be designed to issue privileged SVCs based on the authority code of the user. If this were done, and a programmer with a user authority code (U) attempted to use the fence-sitter, he wouldn't succeed. To be on the safe side, when you write system programs, you should always give the control sections the attributes they need to be able to run; do not rely on your authority code unless all intended users will have an equivalent authority code.

Nonprivileged system programs accessible to user programs have module names that begin with SYS. Analogous to the resident supervisor and privileged programs, control section and entry point names are formed by adding a character to the end of the module name. For instance, SYSABC is an entry point in the nonprivileged system program SYSAB. Names beginning with SYS can be freely referred to by all programs, privileged or otherwise; SYS names can only be defined by control sections with the SYSTEM attribute (see Appendix D for further details).

Nonprivileged system programs not accessible to user programs generally use symbols beginning with CE.

LINKAGE CONVENTIONS

The purpose of a linkage convention is to standardize the method of transferring information and control from one program (the calling program) to another (the called program). Standardization allows the use of system macro instructions for the generation of programs. TSS uses a number of linkage conventions designed to fit a variety of situations while attempting to keep the conventions as similar as possible.

TSS linkage conventions require the calling program to supply a save area for use by the called program. A save area is an area of virtual storage, accessible to the called program, in which it can save the contents of registers, if necessary. Also, a save area contains forward and backward pointers to other save areas, forming a chain. If one program calls a second, and the second program calls a third, the pointers relate the respective save areas. Thus, if you know where one save area is located, you can find the others. The format of the save area used in TSS is shown in Figure 7.

The four basic linkage conventions followed by TSS programs residing in virtual storage are summarized in Figure 8. Note that type-1 has a variation (type 1M), making five ways to link programs. These are the only linkage conventions in use among virtual storage program in TSS. All TSS programs are constructed to receive or transfer control, using one of these linkage types. To add to or modify TSS programs you must use these linkage conventions.

In general, TSS system programs use macro instructions for linking programs. Some macro instructions generate more than one type of program linkage; for example, GETMAIN can generate either a type-1 or a type-2 linkage.

The called program frequently does not know which linkage type was used to transfer control to it; usually, it does not need to know. There are exceptions which are covered later. The linkage type must be known by the calling programs, since it is the calling program that supplies the linkage instructions, save area, and register contents.

```
CHASAV  DSECT       FORMAT OF STANDARD 19-WORD SAVE AREA
        DS    0F    ALIGN ON WORD BOUNDARY
SAVLEN  DC    1F'76'  LENGTH OF SAVE AREA AND APPENDAGES IN BYTES
SAVBPT  DS    1F    BACKWARD POINTER. ADDRESS OF SAVE AREA, IF ANY,
*                     USED BY CALLING PROGRAM
SAVFPT  DS    1F    FORWARD POINTER. ADDRESS OF SAVE AREA, IF ANY,
*                   SUPPLIED BY USER OF THIS AREA TO PROGRAMS IT CALLS
SAVR14  DS    1F    USED BY CALLED PROGRAM TO SAVE GPR 14
SAVR15  DS    1F    USED BY CALLED PROGRAM TO SAVE GPR 15
SAVR0   DS    1F    USED BY CALLED PROGRAM TO SAVE GPR 0
SAVR1   DS    1F    USED BY CALLED PROGRAM TO SAVE GPR 1
SAVR2   DS    1F    USED BY CALLED PROGRAM TO SAVE GPR 2
SAVR3   DS    1F    USED BY CALLED PROGRAM TO SAVE GPR 3
SAVR4   DS    1F    USED BY CALLED PROGRAM TO SAVE GPR 4
SAVR5   DS    1F    USED BY CALLED PROGRAM TO SAVE GPR 5
SAVR6   DS    1F    USED BY CALLED PROGRAM TO SAVE GPR 6
SAVR7   DS    1F    USED BY CALLED PROGRAM TO SAVE GPR 7
SAVR8   DS    1F    USED BY CALLED PROGRAM TO SAVE GPR 8
SAVR9   DS    1F    USED BY CALLED PROGRAM TO SAVE GPR 9
SAVR10  DS    1F    USED BY CALLED PROGRAM TO SAVE GPR 10
SAVR11  DS    1F    USED BY CALLED PROGRAM TO SAVE GPR 11
SAVR12  DS    1F    USED BY CALLED PROGRAM TO SAVE GPR 12
SAVPCT  DS    1F    R(ENTRY POINT) THIS IS SET BY THE CALLING
*                   PROGRAM BEFORE TRANSFERRING CONTROL AND
*                   POINTS TO THE CONTROL SECTION IN WHICH THE
*                   CALLED SYMBOL IS DEFINED AS AN ENTRY POINT
```

Notes:

1. If a program is called and, in turn, calls another program, it must, upon receiving control, establish its own save area, save the address of the calling program's save area in the second word of its own save area and save the address of its own save area in the third word of the calling program's save area.

2. Field SAVPCT contains the R-con of the called program's entry point. This R-con may or may not be an address of a PSECT. If the called program was assembled without prototype control sections (PSECTs), the control section containing the ENTRY statement for the entry point being used by the calling program will be the control section pointed to by SAVPCT. See IBM Time Sharing System: Assembler Language, GC28-2000, for more details concerning R- and V-type address constants.

Figure 7. Format of the standard save area

Type	Transfer Control via	Save-area Format	Parameter Registers (maximum)	Entry-Point Address in Register *	Return Address in Register	Save-Area Address in Register	PSECT Address in Register **
1 (normal)	BASR	Standard	GPR 1	GPR 15	GPR 14	GPR 13	N/A
2	SVC 121 (ENTER)	Standard	GPR 0-1	GPR 15	GPR 14	GPR 13	N/A
1M/2	BASR or SVC 121 (ENTER)	Standard	GPR 1	GPR 15	GPR 14	GPR 13	N/A
3	CALL (CZCJL) Leave-Privilege	Standard	GPR 1	GPR 15	GPR 14	GPR 13	N/A
4 (restricted)	BASR	None	GPR 0-6	GPR 15	GPR 14	None	GPR 13

* Also used for return code, if any.

** Very often, but not always, the PSECT and the save-area address are the same.

Figure 8. Virtual program linkage conventions

TYPE-1 LINKAGE

Type-1 (in some publications, shown as "type-I") linkage is used for transferring control and information between two programs of the same privilege. A nonprivileged program may not use type-1 linkage to call a privileged program; a privileged program may not use type-1 linkage to call a nonprivileged program. Type-1 linkage involves these conventions:

- Using the standard save area
- Using specific registers for designated functions
- Using the Branch-and-Store instruction for the transferring of control
- Preserving registers.

Use of the Save Area

You will find it helpful to refer to Figure 7 for the following discussion. Whenever a program uses type-1 linkage to call another program, the calling program must supply a save area for use by the called program. Before transferring control to the called program, the calling program puts certain information, if applicable, into the save area. The calling program is required to preset some fields of the save area; it may preset others. The first word of the save area (SAVLEN) must contain the length, in bytes, of the save area (minimum 76 bytes) and any appendages to it. The 19th word of the save area (SAVPCT) must contain the R-type address constant (R-con) of the entry point to which the calling progam is transferring control.

The R-value is the address of the control section in which the entry point is defined. The technique in TSS for associating a modifiable control section with a nonmodifiable, reenterable control section is to place the definition of the entry point to the reenterable control section (the ENTRY statement) in the control section that is to be modifiable (usually a prototype control section). Thus, the R-type address

constant for a symbol which is an entry point to the reenterable part of a program provides the address of the modifiable part of that program. In addition to naming an entry point, an R-con can be established for a control section name or an object module name. If a control section name is used, the R-value points to the beginning of the control section. If a module name is used, the R-value points to the first prototype control section contained in the module. If the module does not have any prototype control sections, the R-value points to the first nonprototype control section.

One other field that must be set by the calling program, prior to transferring control to the called program, is the second word (SAVBPT) of the save area. This field is a pointer to a save area used by the calling program when it called. The calling program may not be using a save area, though, and this field may contain zero. If this field does not contain zero, the called program may assume that it points to the save area being used by the calling program.

All other fields of the save area may contain anything. The called program should not assume that fields other than the length field, the R-con field, and the backward-pointer field contain meaningful information.

After receiving control, the called program must save the contents of all the general registers, except register 13, in the save area. At the time the called program receives control, register 13 contains the address of the save area. The other registers are stored in the save area by using register 13 as a base address; e.g., STM 14,12,12(13) saves all the general registers, except 13, in the proper locations of the save area pointed to by register 13. If the called program wishes to use register 13 for its own purposes, it must save register 13 in the backward pointer of its save area. If the called program is going to call another program, and is going to provide a save area for that program, it must store register 13 in the second word of that save area. In this instance, register 13 serves as the backward pointer. Optionally, the user can store register 13 someplace else, not in a save area. If you save register 13 in a save area that you make available to another program, you depend on that program not to write over the save area. If that program is unreliable, you might want to save register 13 in an area accessible to your program alone. That precaution will enable you to restore the registers regardless of what the program you call does to the save area you provide.

The called program does not need to save and restore the floating-point registers. If the contents of the floating-point registers are to be preserved, it is the responsibility of the calling program to save their contents and the contents of its interruption mask.

Contents of the General Registers

Registers 13, 14, and 15 must be preset by the calling program. Register 13 must contain the address of the first byte of the save area that the calling program is providing for the called program. This address must be on a fullword boundary; that is, the two low-order bits of the address must be zero. Register 14 must contain the address to which control is to be returned by the called program. Register 15 must be set to contain the address of the entry point in the called program.

A number of macro instructions can be used to generate type-1 linkages to specific programs. Examples of these macro instructions are CALL, GET, PUT, OPEN, and CLOSE. In using CALL, you specify the name of the program to which you want to transfer control; in using GET, however, the name of the program to which control is to be transferred is supplied by the macro instruction.

The called program always uses register 15 as a return-code register, if return codes are applicable. If a parameter list is passed in register 1, there must be an understanding between the called and calling programs as to its content. In variable-length lists, the word preceding the first word of the parameter list contains a count of the number of parameters in the list. Each following entry is the address of a parameter that has been prestored. Note that in command system routines, a variable-length list is almost always assumed.

Transfer of Control

A type-1 linkage always causes control to be transferred from the calling program to the called program by using the Branch-and-Store (BASR) instruction. Specifically, the resident supervisor is never used to assist in transferring control (no interruption occurs), since the calling and the called programs have the same privilege.

Register 1 may be preset by the calling program with the address of a parameter list. Register 1, 13, 14, and 15 are the only registers used by type-1 linkage.

The CALL macro instruction should be used to generate a normal type-1 linkage. The use of CALL is discussed in Assembler User Macro Instructions.

EXAMPLE: A program transferring control to another program via a type-1 linkage might use these instructions:

```
         L       13,=A(SAVEREA)     LOCATION OF SPACE FOR STANDARD
*                                   SAVE AREA
         USING   CHASAV,13          INDICATE FORMAT
         L       6,=R(SUBR)         GET R-CON OF CALLED PROGRAM
         ST      6,SAVPCT           STORE R-CON IN SAVE AREA
         L       15,=V(SUBR)        GET ADDRESS OF ENTRY POINT
         L       1,=A(PARLIST)      SET POINTER TO PARAMETER LIST
         BASR    14,15              PUT RETURN ADDRESS IN GPR 14
*                                   AND BRANCH
```

The program receiving control might use these instructions:

```
XYZ      PSECT
         ENTRY SUBR          MAKE NAME SUBR EXTERNAL
ABC      CSECT READONLY
SUBR     STM 14,12,12(13)    SAVE ALL REGISTERS
LR       14,13
L        13,72(13)
ST       13,8(14)
ST       14,4(13)
```

to save the general registers and establish definitions for the V-cons and R-cons of the name SUBR. When its processing is finished, the program SUBR might do this:

```
EXIT     LM 14,12,12(13)     RESTORE REGISTERS
         LA 15,4             SET RETURN CODE 4
         BR 14               RETURN TO CALLING PROGRAM
```

TYPE-2 LINKAGE

Type-2 linkage is used when the calling program is nonprivileged and the called program is privileged. All programs designed to be called via type-2 linkage run in the privileged problem state. Type-2 linkage involves these conventions:

- Using the standard save area
- Standardizing the content and use of the general-purpose registers
- Standardizing the method of transferring control
- Preserving registers.

The Save Area

The standard 19-word save area is used in type-2 linkage (see Figure 7). Unlike type-1 linkage, the calling program does not provide this save area. Instead, it is provided by the task monitor, which translates the type-2 linkage into what appears to the called program to be a modified type-1 linkage. The transfer of control from the calling to the called program is through the supervisor when type-2 linkage is used. Coding contained in the task monitor is, therefore, an integral part of the linkage.

Before passing control to the called program, the task monitor initializes a save area for the called program's use. The length field (SAVLEN) contains a byte count of 76 (decimal); the backward pointer (SAVBPT) is zero. The last word of the save area (SAVPCT) contains the R-con of the entry point of the called program. All other bytes of the save area are unpredictable. All programs designed to be called via type-2 linkage can assume that the save area pointed to by register 13 is arranged in this way.

Content and Use of the General Registers

Type-2 linkage conventions assign special functions to registers 0, 1, 13, 14, and 15. The calling program is responsible for presetting registers 0, 1, and 15. The calling program loads register 0 with a parameter that is not an address or, occasionally, the address of a system control block; it loads register 1 with a non-address parameter, or a parameter list address, or the address of a system control block; it loads register 15 with a code, called an ENTER code. The ENTER code identifies the program to be called as would an SVC code in other systems. When control is returned to the calling program, the contents of registers 2 through 14 will be unchanged. Registers 0 and 1 may be used by the called program for returning results. If the called program supplies a return code, it must use register 15.

The task monitor saves all the general-purpose and floating-point registers in its own save area; the task monitor builds a save area for the called program's use, as described in the previous section. The task monitor sets a pointer to this save area in register 13. The contents of registers 0 and 1 are set as received from the calling program. Register 15 is set to the address in the called program to which the task monitor will transfer control. This address is determined by the task monitor, using the ENTER code that was in register 15 when control was received by the task monitor. Register 14 is set to the address in the task monitor to which control is to be transferred by the called program when it has executed. The contents of registers 2 through 12 are unpredictable; they should not be assumed, by the called program, to be significant.

The called program must save the contents of the general registers, since the task monitor requires the contents of the registers passed to the called program to remain unchanged. The called program must return control to the address in register 14. The called program may put a return code in register 15; it may put results in registers 0 and 1. Registers 0, 1, and 15 will be passed back to the calling program as they are received from the called program when it returns control to the task monitor.

Transfer of Control

The calling program transfers control to the called program by issuing SVC 121.

An SVC 121 can be generated by issuing the ENTER macro instruction. SVC 121 passes control through the task monitor to the called program. Most of the time ENTER is used as an inner macro instruction. For example, the macro instruction GETMAIN generates an ENTER if the program in which GETMAIN is issued has been declared by the programmer to be nonprivileged (DCLASS USER). All programs that transfer control via SVC 121 must adhere to type-2 linkage conventions.

EXAMPLE: Assume that you want to get 76 bytes of virtual storage, possibly for use as a save area; you might code it like this:

```
SR      1,1       SET OPTIONS:  NONPRIVILEGED, VARIABLE, BYTE
LA      0,76      BYTE COUNT 76
LA      15,48     ENTER CODE 48 -- GETMAIN (BYTE)
SVC     121       TRANSFER CONTROL
```

Control will be returned to the instruction following the SVC after GETMAIN has been executed. If you had wanted to use the macro instruction GETMAIN, you could have written, GETMAIN R,LV=76 which would have generated equivalent (prefered) instructions.

TYPE-1M/2 LINKAGE

Type-1M/2* linkage applies only to called programs that can be called via both type-1 and type-2 linkages. Programs called by type-2 linkage are always privileged programs. The calling program, however, may be privileged, in which case a modified type-1 (type-1M) linkage is used (with both registers 0 and 1 usable as in type-2 linkages); or the calling program may be nonprivileged, in which case type-2 is used. Since the task monitor makes all type-2 linkages appear, to the called program, as type-1, the called program ordinarily is not affected by the privilege of the calling program.

If a privileged program is being called via type-1M/2, it may need to determine the privilege of the caller. It can do this by comparing the return address in register 14 to the address of the point in the task monitor to which control is returned when a type-2 linkage has been used. For example:

```
CL    14,=V(CZCJER)     COMPARE GPR 14 TO TYPE-2 RETURN
BE    NPCLLR            IF EQUAL, CALLER IS NONPRIVILEGED
 .                      IF UNEQUAL, CALLER IS PRIVILEGED AND
 .                      TYPE-1 LINKAGE IS USED
 .
```

If the calling program is nonprivileged, the privileged called program must issue a CKCLS or a PIREC macro instruction on all addresses passed to it from the nonprivileged routine to insure that it won't change a privileged address for the nonprivileged program. The calling program uses either a type-1M or a type-2 linkage as described previously; if the called program can be called by either of these linkage types, it is using type-1M/2. The calling program treats this linkage as described under type-2.

---

*The use of M with type-1 linkage indicates that register 0 may also be used as a parameter register - in addition to register 1. Register 0 may contain a parameter or a pointer to a system control block.

TYPE-3 LINKAGE

Type-3 linkage is used when the calling program is privileged and the called program is nonprivileged. All programs designed to receive control by a type-3 linkage are designed to run in the nonprivileged state. Type-3 linkage involves standardizing:

- The save area
- The content and use of the general-purpose registers
- The method of transferring control
- Preserving registers.

The Save Area

Type-3 linkage requires the standard 19-word save area; however, the save area is not supplied by the calling program. It is supplied by the Leave Privilege subroutine of the task monitor. The calling program calls the Leave Privilege subroutine, which supplies and initializes a save area for use by the called program (see Figure 7). The Leave Privilege subroutine establishes a 19-word save area which is not read- or write-protected; the nonprivileged called program can gain access to it. The Leave Privilege subroutine sets the first word (SAVLEN) equal to 76. The last word of the save area (SAVPCT) is loaded with the R-con of the called program's entry point. The calling program supplies this R-con to the Leave Privilege subroutine which inserts it into the save area. The remaining 17 words are unchanged.

Content and Use of General-Purpose Registers

Type-3 linkage standardizes the use of registers 0, 1, 13, 14, and 15; the contents of the other registers are as they were at the time the privileged program was entered. The contents of the other registers will be returned, intact, to the calling program. Registers 0 and 1 are used as in type-2 linkages. These registers are passed to the called program as received by the Leave Privilege subroutine from the calling program. The Leave Privilege subroutine loads register 13 with a pointer to the save area it is supplying for the called program. It loads register 15 with the address of the entry point in the called program to which it will transfer control. Then it loads register 14 with the address of an SVC 120 (RSPRV) instruction, which is in the part of the interruption storage area (ISA) that nonprivileged programs can read and write.

Transfer of Control

Control is transferred from the calling program to the Leave-Privilege subroutine with the standard type-1 linkage. Control is transferred from the Leave Privilege subroutine to the called program by an SVC 254 (LVPSW). The use of type-3 linkage always results in an SVC interruption.

EXAMPLE: Within a privileged program, if you want to call a nonprivileged subroutine, you might write:

```
            L       13,=A(SAVEAREA)   GET ADDRESS OF SAVE AREA FOR LVPRV
            L       14,=R(CZCJLE)     R-CON OF LEAVE-PRIVILEGE SUBROUTINE
            ST      14,72(13)         PUT R-CON IN 19TH WORD OF SAVE AREA
            LA      1,PARAMTRS        PARAMETER LIST INTO GPR 1
            L       15,=V(CZCJLE)     ENTRY POINT OF LEAVE-PRIVILEGE
*                                     SUBROUTINE
            BASR    14,15             TRANSFER TO LEAVE-PRIVILEGE SUBR
PARAMTRS    DC      A(ADCONS)         POINTER TO V- AND R-CONS
            DC      A(PARAM1)         POINTER TO PARAMETER 1
            DC      A(PARAM2)         POINTER TO PARAMETER 2
ADCONS      DC      V(CALLED)         ENTRY POINT OF CALLED ROUTINE
            DC      R(CALLED)         R-CON OF ENTRY POINT
PARAM1      DC      F'0'              PARAMETER 1
PARAM2      DC      F'54'             PARAMETER 2
```

The Leave Privilege subroutine will get space to set up a save area for use by the called program. It will load parameters one and two into general registers 0 and 1. It will set up registers 13, 14, and 15, as described previously, and transfer control to the called program via an SVC 254 (LVPSW).

When the called program has been executed, it might return control like this:

```
            LA      0,RESULT1         RETURN OF RESULTS
            LA      1,RESULT2         TO CALLING PROGRAM
            BR      14                RETURN TO CALLER
```

General register 14 points to an SVC 120 (RSPRV) which will cause the Restore Privilege routine to be entered. The Restore Privilege routine will restore the calling routine's original register contents, without disturbing registers 15 (the return code register), 0, and 1 (the result registers).

TYPE-4 (RESTRICTED) LINKAGE CONVENTIONS

Type-4 linkage is used by TSS programs under restricted circumstances for the sake of linkage efficiency. It is found principally in the coding of the language processors.

Type-4 linkage may be used between two programs if all these conditions are met:

- Both the called and the calling programs use the same prototype control section (PSECT).
- The values of address constants required for the linkage have already been supplied by the dynamic loader.
- The called program is not designed to accept type-1, -2, or -3 linkage at the same entry point to be used for type-4.
- Both the called and the calling programs have the same privileges.

Type-4 linkage conventions standardize the use of the general registers and the method of transferring control. There is no standard save area for type-4 linkage.

Use of the General Registers

Registers 0 through 5 are used by type-4 linkage as parameter registers or as pointers to parameter lists. These registers may be used by the calling program to supply information to the called program, or by the called program to return information to the calling program. In

general, the calling program must not assume that the contents of any of these registers will be returned intact by the called program. It is the responsibility of the calling program to load the address of the common PSECT into register 13 before transferring control to the called program. The calling program must set, in register 15, the address of the entry point to which it will transfer control; the address to which control is to be returned is set in register 14. The called program uses register 15 as a return code register, if applicable. The contents of registers 6 and 7 are immaterial; neither program should make assumptions about the contents of these registers. Registers 6 and 7 need not be saved by the called program.

The contents of registers 8 through 12 must be saved by the called program if the called program changes them. The calling program may establish any of registers 8 through 12 as common registers; the calling program may do this only if it has not been called by a type-4 linkage. A common register is a register whose function is understood similarly by the calling and the called programs. If the function performed by a common register, such as pointing to a control block, is required by the called program, the called program may assume that the contents of the common register can be used, as mutually understood between the calling and the called programs. The function of common registers must remain constant in all programs called, in turn, by the called program; their functions must be returned intact to the calling program. The designation of common registers and the nature of their contents are not part of this convention; the use of common registers are an understanding between the calling and the called programs.

Transfer of Control

Control is transferred to the called program by using the instruction:

```
BASR 14,15
```

When using type-4 linkage, interruptions must be masked off; if an interruption occurs, an unrecoverable system error will result.

LINKAGE CONVENTION COMMENTS

This discussion omits type-4 linkage, which is found principally in the TSS assembler, FORTRAN and PL/I compilers, and the linkage editor. Type-4 linkage is used to minimize the overhead associated with program linkage by capitalizing on certain situations that occur in those programs.

We can look at program linkages in two ways: the calling program is the activator; it organizes the linkage information and transfers control. The called program has a more passive role; it receives control and assumes that the linkage information has been organized according to the rules. For some linkage types, a program is inserted between the calling and the called programs; this program performs some of the duties normally associated with the caller. In type-2 linkage, the task monitor's Enter routine is interposed between the calling and the called programs; in type-3, the Leave Privilege subroutine is between the calling and called programs.

From the viewpoint of the called program, most callers look the same. Type-1 linkage doesn't use register 0; the other linkage types may. This is the principal difference from the called program's viewpoint. The called program may return the contents of register 0 to the caller when type-1 is used; for the others, the contents of register 0, if not meaningful, can be ignored. Because of this similarity of appearance to the called program, many called programs can be written in much the same

way. For instance, the SAVE macro instruction can be used to save the contents of the registers in the standard save area supplied by the calling program, and the RETURN macro instruction can be used to restore the registers, load a return code, and return control to the caller. The macro instructions SAVE and RETURN apply to types-1, 1M, 2, and 3 linkage.

FENCE-SITTERS

There are a number of programs in TSS that have no built-in privilege; these programs assume the privilege of the calling program. They are called "fence-sitters."

Linkage to Fence-Sitters

Fence-sitters can be called through a type-1 linkage by either a privileged or a nonprivileged routine. These routines are assigned a hardware storage protection key that makes them read-only to nonprivileged routines. Whenever a type-1 linkage is performed, the PSW protection key is unchanged. Therefore, when called from a nonprivileged program, a fence-sitter routine is a nonprivileged routine. Whenever a fence-sitter service routine is called from a privileged routine, the PSW protection key is zero, and the fence-sitter becomes a privileged routine. This convention is established to efficiently transfer control to those system service routines that link to other (privileged) service routines infrequently.

Some fence-sitter routines have initial entry point names beginning with the letters SYS. This distinguishes them from service routines that must be linked to from a nonprivileged routine through the ENTER mechanism. Other fence-sitter routines are linked to from macro instruction expansions which use address constants which were filled into a data control block by a privileged access method routine.

Writing a Fence-Sitter

Fence-sitters must be constructed carefully. If a nonprivileged program is using a fence-sitter and the fence-sitter is interrupted, it is quite possible that a privileged program will use the fence-sitter during the period of interruption. The fence-sitter must, therefore, be reenterable within the task. Programs may be reentered between tasks or within a task. The use of a prototype control section (PSECT) enables different tasks to use the same read-only control section. Within a task, however, a program is generally made up of one nonprototype control section (which may be shared with other tasks), and one prototype control section (which is never shared with other tasks).

Interruptable service routines, to be reenterable within a task, use multiple save areas and dynamically allocated virtual storage (via GETMAIN).

Fence-sitters can use a number of techniques to prevent the destruction of data if they are interrupted and reentered. Some fence-sitters do not have a PSECT; if they have one, they never modify it. Other fence-sitters require the calling program to supply working storage; still others use GETMAIN to obtain working storage.

Linkage From Fence-Sitters to Other Routines

If a fence-sitter routine needs to link to a privileged service routine, the fence-sitter uses either a type-1 or a type-2 linkage, depending upon the privilege class of the routine that invoked the fence-sitter.

Determining Fence-Sitter Privilege

The fence-sitter can determine the privilege of the calling routine by checking the privilege bit in the VPSW contained in field ISACVP. Parameters can also be supplied by the calling program to tell the fence-sitter what privilege it has. The fence-sitter can also determine the privilege of the calling program by using other information supplied by the calling program, such as the data control block (DCB).

For example, TSS QSAM is designed as a fence-sitter, and will run in the same privilege status as the routine that invokes it. Since it is most often invoked by the problem program, it usually runs in the privilege of the user and may or may not be of the same privilege as the BSAM modules that it invokes. All the BSAM modules invoked, except NOTE, are privileged routines. As NOTE is also constructed as a fence-sitter, and will take on the privilege status of QSAM whenever it is invoked, type-1 linkage is always established to invoke NOTE.

Before establishing linkage to any of the other BSAM modules, it is necessary to determine the status of QSAM subsections. QSAM routines perform this function with respect to their BSAM counterparts by testing the first bit of the VPSW in the ISA. If QSAM is privileged, type-1 linkage is established, using the address constants defined within the data control block. If it is not privileged, type-2 linkage is established via the ENTER SVC.

VIRTUAL MEMORY LOCKING

Rationale

To provide inter-task serialization (that is, the assurance that only one task at-a-time can alter a control block), most shared virtual memory control blocks include space for a Lock Byte, commonly known as a "VM Lock". Any task which has need to alter the block or to read the block without interference can "set the lock", perform its function, and "open the lock"; no other task which observes the lock protocol can interfere.

The "Test and Set" (TS) instruction is used to set the lock -- bit 0 (X'80') on indicates "set". The TS instruction provides inter-processor and inter-task serialization and sets the byte to X'FF'. The lock may be opened by doing: MVI lock,X'00'.

The TS instruction sets the condition code so that the status of the lock, at the instant of the set attempt, may be determined. If the lock is found to be set (by another task), the task wanting to lock must wait until the lock is opened. The waiting process must include retry of the TS instruction.

To provide the resident supervisor with information about tasks which are waiting for VM Locks, a special use of the "Time Slice End" (TSEND) supervisor call macro has been defined. This permits adjustment of task scheduling as well as detection of excessive waiting time. If a lock appears to be "frozen", the task(s) waiting on it will be given a task program interrupt.

To provide the task monitor program with information about lock activity in a task, a one-byte counter ISAVLKCT and a one-bit flag ISAVLK/ISAVLKM have been defined in the Interrupt Storage Area (CHAISA) control

block. These also provide information useful to debugging of lock-related problems.

A system routine, CZACS, is available and should be used to provide release of VM locks in the event of abnormal termination of a task. Refer to "Releasing Interlocks at Abend" in Section 4.

The xxxVLOCK macros provide a consistent means of observing proper VM Lock protocol and simplify program coding, especially in the case of multi-level and/or chained control blocks. All VM Lock processing must use these macros to ensure proper protocol. The only exception is a small set of Virtual Access Method locks which are handled only via modules CZCOH and CZCOI.

Overview

There are six macros in the xxxVLOCK set: CHGVLOCK, CLRVLOCK, LOGV-LOCK, OPNVLOCK, SETVLOCK, and TSTVLOCK. A brief description of each is as follows:

"LOG" generates a control block in which the other five macros record lock status.

"SET" sets a specified VM Lock and records it in a specified LOG.

"OPN" opens the VM Lock recorded in a specified LOG.

"CHG" exchanges two LOGs, to facilitate processing of chained control blocks with individual locks.

"TST" interrogates a LOG to determine the status of the recorded VM Lock.

"CLR" opens the VM Lock recorded in a specified LOG, in a special manner related to ABEND Interlock Release.

The following definitions apply to this discussion of VM Locks:

Tree: a set of two related but different control blocks organized in two logically-distinct "levels". The first level is one control block. The second level is generally a detail expansion of the first and may be a single control block but is usually a list or chain. The specific address of at least the first member of the second level is usually recorded in the first level block and probably varies over some short period of time. The address of the first level is often fixed at system startup or task logon

List: a set of two or more copies (same layout, different content) of a control block which occupy consecutive locations in virtual memory. Any one can be referenced given the address of the first, the length of each (usually fixed), and the number of members.

Chain: a set of two or more copies of a control block which occupy arbitrary locations in virtual memory, where each member contains the address of the next member in the logical order. Any one can be referenced given the address of the first. The first level of a tree may be a member of a chain or a list. The second level of a tree usually is a chain or a list.

Using the xxxVLOCK Macros

Consider the following data structure for illustrating the use of the xxxVLOCK macros.

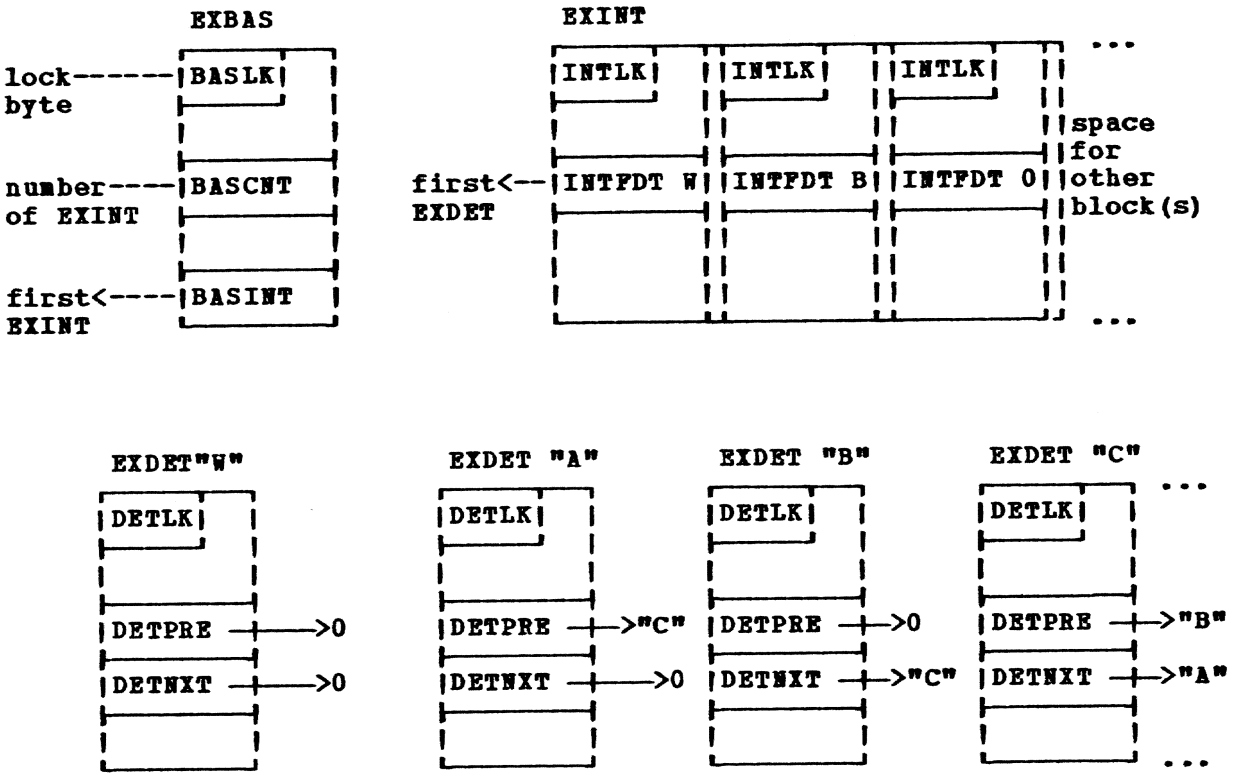


Figure 9. Sample Data Structure for xxxVLOCK Macros

The only constant available to the sample program is the address of EXBAS. System DSECTs are assumed to be available for EXBAS, EXINT, and EXDET.

The objective of the sample program is: to find a particular "class" of EXINT (the second in the example layout); to find the last EXDET; and to add a new EXDET before the last (due to some priority scheme). The number, but not the order, of EXINT may be changed by any task. The number and order of EXDET may be changed by any task.

In the PSECT of a module you may code the following VM Lock Anchor:

```
LBAS   LOGVLOCK   ,   RECORD EXBAS.BASLK
LINT   LOGVLOCK   ,   RECORD EXINT.INTLK
LDET1  LOGVLOCK   ,   RECORD EXDET.DETLK "1"
LDET2  LOGVLOCK   ,   RECORD EXDET.DETLK "2"
LDETA  LOGVLOCK   ,   RECORD EXDET.DETLK "NEW"

.................
```

Pick up the address of EXBAS, providing addressability by the DSECT, and then:

```
SBAS    SETVLOCK  BASLK,LBAS      LOCK EXBAS
```

Next, pick up the address of the first EXINT from BASINT.

After providing addressability for EXINT, you can DROP the addressability for EXBAS (not required for the OPNVLOCK later).

```
QINT    EQU   *
```

Test EXINT to find out if it is the one you want. If it is, go to OBAS. If it is not, check BASCNT to see if there are any more EXINTs. If there are no more EXINTs, go to NINT.

To check the next EXINT, increment your EXINT base register by the length of EXINT, and go to QINT.

```
.................

NINT    OPNVLOCK  LBAS
```

Here, you have run out of EXINT, so you can log a SYSER, or whatever, and end the program.

```
.................
```

Assuming you have found the EXINT you want, you can:

```
OBAS    SETVLOCK  INTLK,LINT      LOCK DESIRED EXINT
        OPNVLOCK  LBAS            UNLOCK EXBAS
```

Then pick up the address of the first EXDET from INTFDT and DROP addressability for EXINT. Now you must setup addressability for EXDET, and

```
SDET    SETVLOCK  DETLK,LDET1     FIRST ("B")

        OPNVLOCK  LINT            UNLOCK EXINT
QDET    EQU   *
```

You are now sitting on an ("B", "C", "A") EXDET, looking for the last EXDET. Check DETNXT and, if it is zero, you are on the last ("A"), so you go to ADET.

Otherwise ("B","C"), needing to go to the next EXDET, you load ("C","A") DETNXT into your EXDET base register, and

```
CLRVLOCK  LDET2               PRIOR (0,"B")
CHGVLOCK  LDET1,LDET2         EXCHANGE ("B"/0,"C"/B)
SETVLOCK  DETLK,LDET1         NEXT ("C","A")
```

Then go to QDET

```
.................

ADET    EQU   *
```

You are now sitting on the last ("A") EXDET, needing to add a new EXDET <u>before</u> this one. Find a slot to build a new EXDET and initialize it to 0.

Now is the time to change the current ("A") DETPRE,

```
      L  PREREG,DETPRE      @ "C" FROM "A"
     ST  NEWBAS,DETPRE      @ NEW INTO "A"
```

Then load your EXDET base from PREREG to cover the previous ("C") EXDET and

```
      L  NXTREG,DETNXT      @ "A" FROM "C"
     ST  NEWBAS,DETNXT      @ NEW INTO "C"
```

Now you are ready to fill in the new EXDET; load your EXDET base from NEWBAS, and

```
     SETVLOCK  DETLK,LDETA      NEW
     OPNVLOCK  LDET1            NEXT*("A")
     OPNVLOCK  LDET2            PRIOR("C")

     ST  PREREG,DETPRE          @ "C" to new
     ST  NXTREG,DETNXT          @ "A" to new
```

Fill in the new EXDET you are on, and

```
     OPNVLOCK  LDETA     NEW
```

Now end the program, which should include:

```
     TSTVLOCK  LDETA,errorDA         LOCK
     TSTVLOCK  LDET2,errorD1         * WAS
     TSTVLOCK  LDET1,errorD2         ** LEFT
     TSTVLOCK  LINT,errorI           *** SET,
     TSTVLOCK  LBAS,errorB           **** ABEND

     ..............
```

Your ABEND Interlock Release Routine should include:

```
     CLRVLOCK  LDETA
     CLRVLOCK  LDET2
     CLRVLOCK  LDET1
     CLRVLOCK  LINT
     CLRVLOCK  LBAS
```

SUPERVISOR CONTROL LOCKS

The system resource locks are maintained in a table in the supervisor (instead of virtual memory) with a general purpose ENQ/DEQ function. This table allows the supervisor to recover any resources held by a task being deleted and to maintain reliable scheduling information.

If these locks were in virtual memory and a task abended or logged off leaving a lock set on a system required resource, the system would have to be reIPLed, because users would be unable to access the required resource. Also, the system would be unable to determine which task was holding the lock.

The implementation is as follows:

1. A hashed and chained dictionary of resource names is maintained in the supervisor.

2. Two macros ENQ and DEQ (refer to the Assembler User Macro Instruction manual) are provided to interface with the new mechanism. The ENQ function is performed asynchronously with the task. The completion, successful or otherwise, is posted back to the issuer in an ECB.

3. The important virtual memory locks use this mechanism -- VAM lock modules CZCOH and CZCOI; also, FIND, STOW, and RCR.

4. The module CZAHC posts an ECB specified by an external interrupt code queued on the task by the resource control module.

VAM LOCKING

VAM's lock modules CZCOH and CZCOI use the ENQ/DEQ mechanism. CZCOH issues an ENQ using the lock's VMA as the resource name and CZCOI issues a DEQ to remove the lock.

The parameter list for CZCOH allows the caller of CZCOH to request a return code and to request that CZCOH abort the lock request if the user attempts to attention out of the request. The lock type code has had the two high order bits (X'80' and X'40') defined to request one of the above actions. If the bits are zero, CZCOH processes as before. If the 0 bit (X'80') is a 1, CZCOH returns a code to the caller signifying the success of the lock request. CZCOH will return one of the following codes:

0 - lock successfully set

4 - wait time expired; register 1 contains the taskid of the lock holder

12 - lock request purged before lock was set

If the 1 bit (X'40')is a 1, CZCOH will test for a pending attention from th user while waiting for a lock to be successfully set. If an attention is detected, the lock request is purged and an 8 code is returned to the caller.

Whenever CZCOH returns with a non-zero code the lock request has not been fulfilled and it is up to the caller to recover from the situation.

To facilitate the wait-time-exceeded case, and to present to the user a standard message with standard inserts, a second entry point has been added to CZCOH -- CZCOH2. If this entry point is called with a suitable parameter list, CZCOH2 will build and return to the caller a message containing the appropriate information to define the unsuccessful lock attempt to the user. The parameter list for CZCOH2 is the same as CZCOH except that two words have been added to the end of the list. Word 5 contains the taskid returned by CZCOH when a lock request has exceeded its wait time. This is assumed to be the taskid of the task holding the requested lock too long. Word 4 is a pointer to a message insert to be used for lock type X'14' calls. It is assumed that this insert explains the type of control block the caller is attempting to set. The address should point to a character string preceded by a one-byte length.

The CZCOH1 parameter list is as follows:

Register 1 -- 3 word list

word 1 - address of lock point
word 2 - address of two or three byte control field
word 3 - address of DCB or 0
word 4 - address of message insert for type X'14' calls (required)

Format of Control Field

byte 1 - type of access requested
C'W' - exclusive access (write)
C'R' - shared access (read)

byte 2 - control block being locked, and flags
codes: X'00' - SDST entry
X'04' - RESTBL
X'08' - POD
X'0C' - member header
X'10' - page lock
X'14' - other

bits 0 and 1 of the code byte are used as flags
0 (X'80') - return with a return code
1 (X'40') - test for pending attentions

byte 3 - option flags for a type X'14' call; this byte is used only on type X'14' calls

bit 0: 0 - system controlled resource
1 - user controlled system resource

bit 1: 0 - normal wait required
1 - lock request is an immediate request; do not wait if already locked

CZCOH1, if requested, will return one fo the following return codes:

0 - successful attempt
4 - wait time exceeded; lock holder's taskid is in register 1
8 - attention detected (only if bit 1 is a 1)
12 - lock request purged

The CZCOH2 parameter list is as follows:

Register 1 -- 5 word list

word 1 - address of lock point
word 2 - address of two byte control field
word 3 - address of DCB or 0
word 4 - address of message insert for code X'14'
word 5 - taskid of lock holder

Format of Control Field

byte 1 - type of access requested
C'W' - exclusive access (write)
C'R' - shared access (read)

byte 2 - control block being locked, and flags
codes: X'00' - SDST entry
X'04' - RESTBL
X'08' - POD
X'0C' - member header
X'10' - page lock
X'14' - other

bit 0 of the code byte is used as a flag
0 (X'80') - do not return; CZCOH will call ABEND

CZCOH2 will return to the caller the length of the build message in register 0 and the address of the message in register 1.

DYNAMIC SCHEDULE TABLE TRANSITION

The ENQ/DEQ mechanism provides another scheduling capability to TSS. This capability allows a task to be given a dynamic schedule table transition as page stealing does.

A second entry point in CEAKZ (CEAKZ2) makes the transition and updates all required system control counts. CEAKZ2 is called with the new schedule table level in register 0 and the TSI address in register 1. CEAKZ2 validates the new level, calculates a second new level if the first new level has the prejudice flag on, and updates the task dispatchable priority from STEPRIOR and the number of quanta left by calculating the number of quanta used from the old level and subtracting that value from the new quanta count. The task is guaranteed at least one additional quantum.

If the task is on the inactive list nothing else is done. If the task is on the eligible list a new scheduled start time is computed and the task is reslotted in the eligible list using the new priority and scheduled start time. If the task is in the dispatchable list, the task is reslotted using the new priority. SYSECB and SYSBTCNT are updated to reflect the possible change in STEMAXCR between the old and new schedule table levels.

Currently, CEANB, PULSE SVC, CHANGE SVC and ENQ/DEQ are the only routines using CEAKZ2.

ENQ/DEQ SCHEDULING

The ENQ/DEQ mechanism uses the Holding Lock exit in the schedule table as its scheduling exit. When a task acquires its first system resource (i.e., holding resource count = 1) the ENQ/DEQ module CEARS uses the Holding Lock value in the current schedule table entry as the next schedule table level for the task. CEARS saves the task's current schedule table index and calls CEAKZ2 to take a dynamic transition to the new schedule table level. When the task frees its last system resource, CEARS retrieves the saved schedule table index and calls CEAKZ2 to return the task to its previous level.

The ENQ/DEQ module schedules only for resource entities called system resources. A system resource is one that is allocated and released completely under control of privileged system control, without ever returning to non-privileged code. Examples are RESTBL locks, POD locks and user table header and entry locks. User controlled resource entities are resources whose duration of allocation is controlled from the non-privileged code or by user action. Examples are dataset access locks, member access locks and page locks. With these resources, the duration of the allocation is usually controlled completely by user action.

The task when waiting for a resource will issue an AWAIT SVC against an ECB. This allows the current system AWAIT extension code and schedule table value to be used. The current ENQ/DEQ mechanism does not use a waiting on lock mechanism; it is not required.

With ENQ/DEQ, the system considers it an error if a task issues a WAIT or TWAIT SVC while holding a system resource. This error currently only results in a minor syserror.

RESOURCE CONTROL FACILITIES

TSS provides facilities to the system programmer for controlling the allocation of system resources to users and for keeping records of resource usage.

The control and accounting functions apply to the following resources: CPU time, terminal time, number of concurrent background tasks, amount (in pages) of permanent and temporary public storage, number of direct access devices, magnetic tape drives, high speed printers, readers and punches, and the number of records read and written by BULKIO.

Accounting Overview

An overview of the accounting facility's logic is given below. Resources are allocated and kept track of by means of several tables (the user limits table, the system user table, the active user list table, and the task accounting table) and several macro instructions and commands: USAGE, UPDTUSER, JOIN, and REJOIN. USAGE and UPDTUSER are commands as well as macro instructions; JOIN and REJOIN are commands only. The function of each table used by the accounting facility is summarized below. The macro instructions are described in Part II or in Assembler User Macro Instructions. The JOIN and REJOIN commands are described in Manager's and Administrator's Guide. The USAGE command is described in Command System User's Guide.

User Limits Table (SYSULT) - DSECT CHAULT

SYSULT is a table containing the maximum amounts of resources a user is allowed at one time or over a given period. Depending on the resource (CPU time, I/O devices occupied, etc.), the time may be the duration of a task or the duration of some installation-established accounting period. Initially, when the system is supplied, two sets of limits are defined in the table: one set for system programmers and one set for user programmers. The limits supplied with the system for each of these two sets are summarized in Appendix G.

When a user is joined to a system, the RATION operand of the JOIN command determines which set of limits the user has. The set of limits is recorded in the user's entry in the system user table (see below).

An installation may change the system-supplied values for either set of limits; it may also define and add to the user limits table up to seven other sets of limits, which can be identified by additional RATION keys when JOIN is issued.

System User Table (SYSUSE) - DSECT CHAUSE

The primary purpose of SYSUSE is to provide a list of all legal TSS users, with their attributes or characteristics, for reference by system routines. It contains an entry (or record) for each user, referred to as a user table entry (UTE). In addition to the user's ID, password, privilege class, authority code, etc., each UTE also contains fields for recording accounting data about the user's resource usage. This data is accumulated from the time he is joined to the system till he is quit. It holds accounting statistics for resource amounts being used and resource products (that is, amounts multiplied by time used). At JOIN

time a user's limit rations (from SYSULT) are also copied into his entry in SYSUSE.

Active User List Table - DSECT CHAAUL

This table contains an entry (or record) for every task and one for each user whose data sets are being shared by that task. These entries are created by a call to the Resource Control Routine (RCR) CZCUA. The entries are referred to by task identification (task ID) and user identification (user ID). Several entries may be associated with one task: the primary entry (based on a user's task ID and user ID), an entry for each shared data set opened by the user (based on the user's task ID and the owner's user ID), and a system entry based on task ID and the TSS*** ** user ID. Each entry contains a temporary record of a task's use of system resources from the beginning to the end of that task.

Task Accounting Table - DSECT CHAACT

This table is a work area in which a task's resource statistics (pertaining to its primary entry in CHAAUL) are recorded when that task comes to an end. This work area is made available to user-written accounting routines at the end of each task.

Resource Control Operation

To get the system started, two entries (two sets of limits) have been defined in the user limits table (SYSULT). An installation, however, may provide new sets of limits or change existing ones at any time (see "Retrieving and Modifying Accounting Data Sets" also in this section). When a user is joined to the system, he has an entry (the UTE) created for him in the system user table (SYSUSE). At that time, the user's limits (from SYSULT) are automatically copied into that UTE.

As each user issues LOGON, the UTE for that user is copied from SYSUSE into a shared virtual storage table, or in cases where another task is active for the same user ID, the task is connected to the UTE already copied into shared virtual storage. Each task then has several task ID entries assigned to it in a second virtual storage table, called the active user list (CHAAUL). Thus, the two virtual storage tables, defined by DSECTS CHAUSE and CHAAUL are used to record user ID and task ID accounting data, respectively. During the execution of a task, the task can refer directly to these virtual storage tables and update user ID and task resource statistics in those tables without interfering with their use by other tasks belonging to the same user.

Dynamic Accounting

During task execution, every time a user requires additional resources, the system checks the amount of needed resource plus his currently-used amount against the user's limits to make sure he has not exceeded them. If they are not exceeded, the resources are allocated to him and the virtual storage accounting table (defined by CHAUSE and CHAAUL) is updated to reflect the additional usage. This process occurs whenever external pages are assigned to, or relinquished from, a user's VAM data sets, or when devices (such as disks, tapes, printer, and reader-punches) are assigned or released. CPU time and connect time are not updated dynamically; they are updated only at the end of each task.

The "time last changed" fields in the user's UTE and in each user task's active user list entry are used to calculate the product (resource multiplied by time used) fields in those tables when additional resources are allocated.

Keeping track of resources is accomplished (at LOGOFF and ABEND) by the system accounting subroutine and (dynamically) through use of a call to the Resource Control Routine.

Shared Data Set Accounting

When a task uses a shared data set belonging to another task, accounting statistics are recorded in the appropriate entry in the active user list based on the user's task ID and the owner's user ID. These accounting statistics remain in this table until an RCR call is made for the shared data set. At this time, the accounting statistics pertaining to that task's use of the shared data set are used to update the SYSUSE. The entry in SYSUSE, associated with the user ID for the owner of the shared data set, is updated. Thus, if you share your data sets, you will be charged for the use of the resources associated with those data sets.

Resource Control Routine (RCR), CZCUA

RCR is used to open the accounting tables, compare requested resource usage with the user's limits, dynamically update the product fields in the virtual accounting tables (described by DSECTS CHAUSE and CHAAUL), and release or vacate resources when a task has finished using them.

When a task goes to LOGOFF or ABEND, RCR is called to tabulate accounting data for each user ID and each task or subtask (for example, an express batch subtask), and makes sure all resources used since the last allocation have been updated. The task accounting data tabulated by this process is:

1. Temporary page seconds
2. Permanent page seconds
3. Private disk seconds
4. Private tape seconds
5. Private printer seconds
6. Private reader/punch seconds
7. Total number of auxiliary storage pages
8. Total number of TWAITs
9. Total number of AWAITs
10. Total time slice ends
11. Total page-ins from auxiliary storage
12. Total page-ins from external storage
13. Total page-outs to auxiliary storage
14. Total page-outs to external storage
15. Maximum pages held on auxiliary disk
16. CPU time
17. Terminal connect (CONN) time

CPU time and connect time are updated for conversational tasks. For nonconversational tasks, only CPU time is updated; connect time is reset to 0.

The shared virtual storage image of the user table entry should now reflect the status of the user's system resources. These updated statistics are then recorded in the permanent SYSUSE data set in external storage.

For both conversational tasks and nonconversational tasks or subtasks, the accounting subroutine branches to a dummy task accounting module at CZAGA. Upon entry, this module returns control to the calling routine. It is provided as a hook for user-written accounting routines. A user can replace the dummy module with his own task accounting routine.

At the time of the call, register 1 contains a pointer to a parameter list that contains a pointer to the work area containing the task's accounting statistics. Users who want to examine this data can establish addressability to the work area, by using the DSECT CHAACT. Signed binary values representing each resource are stored in the work area.

When the accounting routine returns to the caller, a subsequent RCR call is made to remove the appropriate task-oriented entry from the active user list and to close out the user table entry in shared virtual storage. This is done by disconnecting the task from the shared-virtual-storage UTE or, if there are no other tasks active for the user, by writing the shared-virtual-storage UTE to SYSUSE and freeing the virtual storage copy.

INSTALLATION ACCOUNTING ROUTINES

An installation may establish its own accounting routine by writing a privileged module, assigning to it the module name CZAGA, and replacing the dummy accounting routine at CZAGA with the installation's module.

IMPORTANT ACCOUNTING CONSIDERATIONS

When the installation accounting program receives control from the system accounting subroutine, it can write to SYSOUT, can define (DDEF), open, and close its own accounting data sets, and can perform most system functions. However, the user should be aware that all non-system data sets have been released; only selected system data sets remain open (for example, SYSUSE, SYSCAT, SYSOUT, and SYSMLF). In addition, the user may be restricted in the functions he can perform in an accounting routine invoked by ABEND, depending on the cause of the abnormal end.

System programmers should be aware that product (resource multiplied by time) fields (except for CPU and connect time) accumulated in the virtual storage copies of SYSUSE (CHAUSE) and CHAAUL are updated as resources are allocated or through use of the USAGE command or macro instruction. Although those tables are valid as of the last update, the work area (defined by DSECT CHAACT), in which task accounting data is accumulated, is updated only at LOGOFF or ABEND. CPU time, connect time, and the permanent copy of the product fields, recorded on SYSUSE, are updated only at LOGOFF or ABEND. A system programmer should also be aware that if the system is taken through an RPS and UPDTUSER sequence, resource counts are updated, but the product fields are reset to zero. Thus, before executing such a sequence, installations should save the existing accounting data for later processing.

Accounting by User or Task ID

Installation accounting routines can associate accounting information with a user ID or a task ID. If the user ID is chosen, the already opened virtual copy of the SYSUSE data set can be examined. The ability to address the user table entry can be established from the address of the UTE found in the task common field TCMVLU. The DSECT CHAUSE can then be used to extract information, and the installation accounting routine can perform any desired arithmetic (all entries in the table contain signed binary data). Accounting by task ID can be accomplished by examining the work area, defined by the DSECT CHAACT, in which task statistics are recorded at the end of each task's processing.

Accounting by Charge Number

If an accounting routine desires to accumulate billings based on charge numbers, it should be aware that TSS accepts all charge numbers without validating them. If a charge number is defaulted at LOGON, the charge number assigned to a user when he is joined to the system is used for that task's accounting at LOGOFF or ABEND. It is the responsibility of the installation's accounting routine to verify charge numbers. If the accounting routine finds a charge number is invalid, it can use the user ID.

Accounting on a Project Basis

A user who wishes to be charged separately by project may do this by being joined to the system under separate user IDs. In order for him to have access to his data under separate user IDs, he must use the PERMIT and SHARE commands for his various IDs.

DISPLAYING AND ALTERING ACCOUNTING STATISTICS

If at any time during a task a user wants to know how many resources are assigned to his user ID, or what his resource limits are, he can issue a USAGE command or macro instruction to display them on his SYSOUT device. Remember these are user statistics rather than task statistics, except for CPU and CONN time current fields. The USAGE command or macro instruction also updates the product fields in the virtual storage accounting tables (CHAUSE and CHAAUL). A user ID option is available with USAGE for system programmers, managers, or administrators, which displays the usage statistics for any user joined to the system. Similarly, a 'RESET' option can be used by managers and administrators to reset a user's product fields to zero (see the description of the USAGE command in the Manager's and Administrator's Guide).

Managers and administrators can use the REJOIN command to change the user's limits, increasing his resources (see REJOIN in the Manager's and Administrator's Guide).

Note: If a user's resource usage were to change when the system accounting facility was not in operation, his accounting statistics would not be updated (for example, if an RPS and CVV command sequence was performed or in the event of system failure). In such cases, an O authority system programmer should employ the UPDTUSER command or macro instruction to update the resource count statistics in the user table for all users joined to the system (the product fields in the UTE are not updated at this time).

RETRIEVING AND MODIFYING SYSTEM ACCOUNTING DATA SETS

To assist users interested in writing accounting routines and establishing their own resource limits, information on SYSULT and SYSUSE is summarized below. The DSECTs for the system accounting tables (CHAULT, CHAUSE, CHAAUL, and CHAACT) can be found in the ASMMAC system library. Appendix G shows the resource limits supplied by IBM.

SYSULT characteristics:

- VISAM member of TSS*****.SYSLIB.
- Each record in the table, specifying the limits for a specific user or type of user, is 64 bytes long.
- The hexadecimal key, in the first four bytes, is in the range 1-9, inclusive.
- DSECT CHAULT.
- All limit fields contain a fullword of signed binary data.
- USAGE documentation describes the general contents of each field.
- Although the JOIN command indicates that there are only two available sets of limits in the user limit table (SYSULT), additional sets may be added to the system (or existing limits changed) by modifying SYSULT. This can be done by issuing the MODIFY command using the hexadecimal option. For example:

User: modify syslib(0) (sysult),,lrecl=64,keyln=0,rkp=0,recfm=f

System: Either prompts user for the hexadecimal input data or unlocks the terminal keyboard.

User: Enters the new set of limits in the form:
x%00000003000000e10000232b600004e200000100000000c-
8000000020000000200000002000000002%e

For additional information on the MODIFY command, see Command System User's Guide.

SYSUSE characteristics:

- Each user ID record (that is, a UTE) is 256 bytes long.
- Has an EBCDIC key.
- DSECT CHAUSE.
- All accumulation and product fields contain fullword signed binary data.

CREATING YOUR OWN PRIVILEGE CLASSES

You may assign one, or a combination of several, predefined privilege classes (see Appendix H) to a user at the time you join the user to the system. In addition, you can create additional privilege classes, using any of the remaining (undefined) alphabetic characters. To accomplish this, you can use the CLOP macro instruction during system generation to introduce a new privilege class to the system; this records the privilege class in task common (DSECT CHATCM) as a valid privilege class.

Any routines you want associated with the newly defined privilege class must be coded to examine the appropriate privilege class bits in the four class bytes in task common.

At JOIN time, the privilege class you have defined can be assigned to users of your choice; they may then use the set of routines you have made available to them.

ESTABLISHING PRIVILEGED INTERRUPTION SERVICING ROUTINES

For normal returns from a privileged interruption servicing routine, execution resumes at the next sequential instruction following the point of interruption, just as it would from a nonprivileged interruption routine (see "Processing an Interruption" in Assembler Programmer's Guide). However, if a privileged interruption servicing routine wants to modify the return address at which control is to resume, it must use a different process than that used by nonprivileged routines. Rather than modifying the old VPSW in the area pointed to by register 0, as done by nonprivileged routines, the privileged interruption servicing routine must modify the instruction address in the old VPSW at location ISA10P (X'730') in the interrup storage area (ISA). Control then passes from the interruption servicing routine to the newly specified address, rather than to the next sequential instruction.

SCHEDULING TIME BY A SYSTEM TABLE

CPU time in TSS is scheduled by means of a system table (the schedule table, CHASTE) that permanently resides in main storage as a system control block. The supervisor refers to this table when schedulling tasks,

both at task initiation and during execution of the task, to determine when next to schedule the task and for what amount of CPU time. The scheduling table may contain as many as 256 scheduling levels, called schedule table entries (STE).

User priority and task type (that is, conversational or batch) determine a value that is assigned to the STE field of the task's TSI; that value then determines the task's initial scheduling parameters. Once a task has been initiated, the supervisor may move a task to another level, as for example, when a task is switched from conversational to batch mode. Levels are also adjusted for tasks that are to be I/O-bound or execute-bound.

The system programmer may also change the scheduling of a task. The PULSE macro instruction permits you to change the STE level of a task to another pre-set "pulse level" that is associated with the current level. The CHANGE macro instruction permits you to change the task's level to a level that you specify. The PULSE macro instruction is unrestricted; CHANGE is restricted to privileged routines.

The PRESENT macro instruction permits you to determine the schedule level of a task; it is unrestricted.

The values in the scheduling table are established as a function of system generation and maintenance. Full information on the system table is provided in System Control Blocks. System Logic Summary explains how the supervisor uses this table for scheduling. To alter table values, consult System Generation and Maintenance.

DEADLINE DISPATCHER

ACTIVE LIST ORDERING

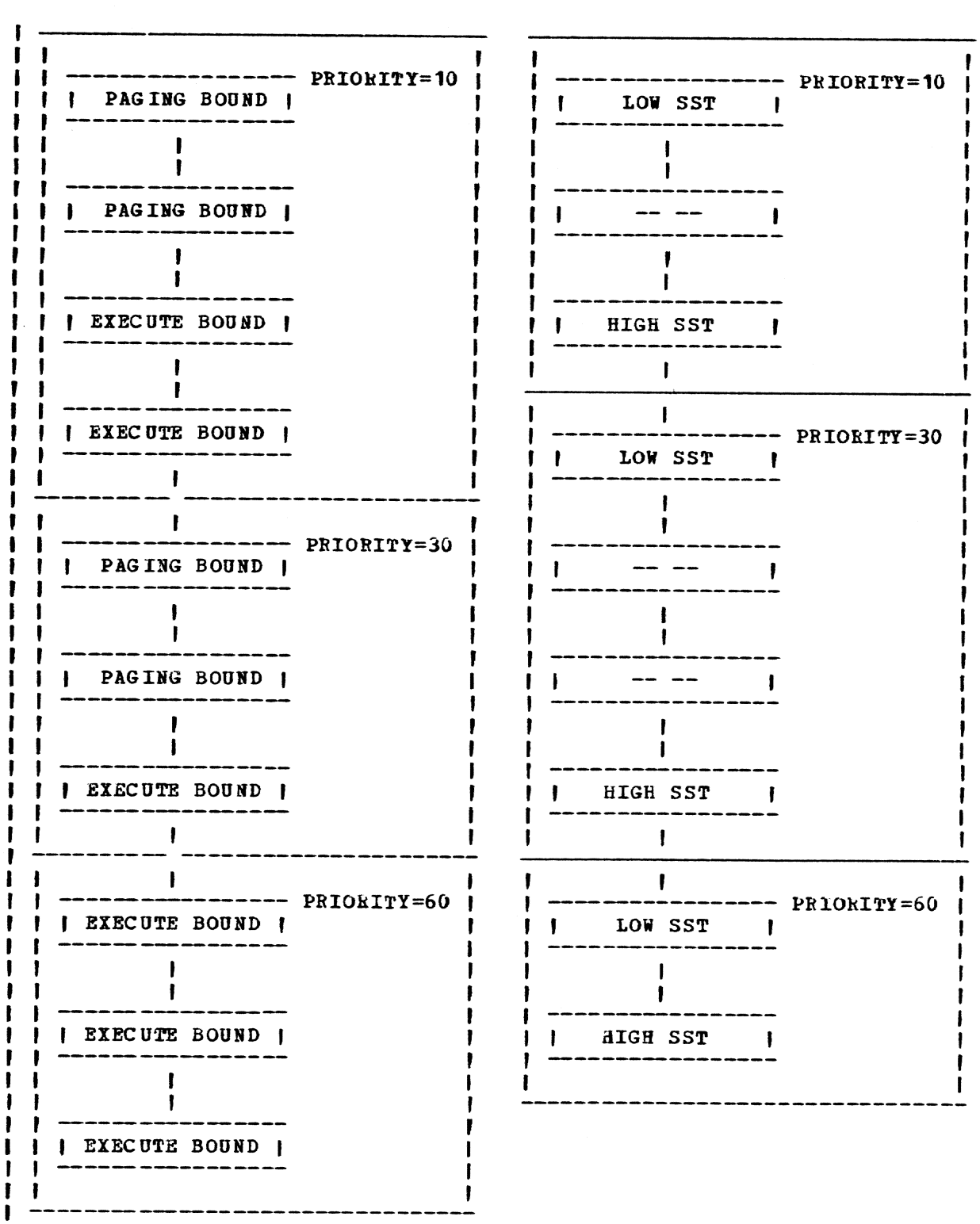
The active list is subdivided into the dispatchable list and the eligible list. The dispatchable list consists of tasks which are in main storage competing for CPU time. The eligible list consists of tasks which are ready to execute but have not yet been brought into main storage.

Tasks on the dispatchable list are ordered by the priority schedule table field (STEPRIOR) with the smallest priority number first. Tasks with the same priority number are ordered so that paging-bound tasks are ahead of execute-bound tasks. Paging-bound tasks are those which, in one quantum, cause more page relocation exceptions than the amount specified in the maximum page relocation exceptions schedule table field (STEMRQ). Tasks are initially placed between paging-bound and execute-bound tasks having the same priority number. At the end of each quantum, a task is reclassified as paging-bound or execute-bound and placed after all other tasks of the same priority number and classification.

Tasks on the eligible list are also ordered by priority (STEPRIOR) with the smallest priority number first. Tasks with the same priority number are ordered by SST (Scheduled Start Time) with tasks furthest behind schedule (i.e., lowest SST) first.

Because the dispatchable list is now ordered by priority, greater care must now be taken in selecting schedule table priority values. It is now possible for a task with a small priority number to completely dominate a CPU and effectively lock out other tasks for long periods of time. Greater emphasis should now be placed on the delta to run schedule table field (STEDELTA) for controlling the eligible list order.

The following diagram schematically shows the active list ordering for a schedule table with three priority numbers.



SCHEDULE TABLE

The Schedule Table header has an entry point CHBSTH; the table is 64 bytes and includes 12 for installation use and 21 that are reserved. All priority fields (STEPRIOR) have been set to X'80'. The delta to run fields (STEDELTA) have been adjusted to maintain the proper ordering of the eligible list.

CPU/APU SCHEDULING

The dispatcher (CEAKD) searches the dispatchable list from the top (highest priority task) to find a task that is ready for execution. If a ready task is not found, the dispatcher will either exit to the wait state or the queue scanner depending on whether or not any supervisor work exists. If a ready task is found and more than one processor exists, the dispatchable list search continues for another ready task that has an expired deadline dispatch time (see the RTTCTL macro instruction description). If a second ready task is found and the other processor is running a lower priority task, the other processor is signaled to reschedule its executing task and search the dispatchable list. The first ready task found is then set in execution.

REAL TIME TASK FLAG

The time slice end processor (CEAKT) will keep all of the task's pages in real storage if the real time task flag (TSIRTT) is on. This flag is turned on and off by the RTTCTL macro instruction. To prevent excessive time slice ends for maximum pages allowed, page stealing should be turned on in the schedule table levels for tasks that use the RTTCTL macro instruction.

ADDRESSING AN I/O DEVICE

The initiation of a virtual I/O operation does not require the use of the interruption storage area in the way the initiation of a real I/O operation requires the use of the prefixed storage area. Virtual channel programs are constructed using I/O request control block (IORCB) and channel command words that are similar to real CCWs (see the discussion of IOCAL). All I/O operations in a task need not use IORCB channel command sequences, though. Some I/O operations, such as Virtual Access Method (VAM) operations, are performed by using two Supervisor Call instructions (see the descriptions of the PGOUT and SETXP macro instructions) that take advantage of the characteristics of the address translator.

Because most I/O devices attached to the system have more than one path to storage, these devices have multiple real addresses. The supervisor's pathfinding program selects the address to be used. To distinguish an I/O device from the path (hardware address) used to gain access to it, each device attached to the system is given a unique number, called the symbolic device address. The assignment of symbolic device numbers is unique at each installation.

In addition to the symbolic device address, some I/O operations require the use of a relative page number. The relative page number is a 16-bit quantity allowing a device to have 65,536 pages. For certain I/O operations (for example, PGOUT), each device is organized into pages. Since each page is 4096 bytes, the position of a given page on all devices of the same type can be determined. Thus, page 136 begins at the same cylinder, track, and record address for all IBM 3350s. In other words, given a relative page number and a device type, it is always possible to figure out where, on that device, the page can be found.

The system symbolic device address and the relative page number make up the external page address; they identify the location of a page on external storage.

TIMEKEEPING FACILITIES

The time-sharing system maintains information about several categories of elapsed time. Timekeeping facilities accumulate statistics that can be used for accounting purposes and for monitoring system and program performance. User programmers can set a time that measures task execution time or elapsed calendar time. System programmers can measure time slices and time intervals during which they may want the CPU, or a task, to be in the wait state. The TSS timekeeping facilities are composed of time cells and the macro instructions that set and read those cells.

TIME CELLS

Time cells are used by TSS to store information about elapsed time, estimated time, and related data. The relationships of the principal timekeeping cells (or fields) are described briefly below under "Timekeeping Operation."

Categories of Time

The information in the time cells provides the system with the two basic types of timekeeping statistics maintained by TSS: task time and realtime. In addition, CPU time is maintained as a part of real-time maintenance (see "Realtime Maintenance"). Realtime is the actual time (in microseconds) that has elapsed since March 1, 1900; from that starting date every year divisable by four is a leap year (366 days).

Timekeeping Operation

The time-slice allocated for a task is specified in its current schedule table entry. Each time-slice is composed of one or more quanta. The number of quanta and the number of microseconds in each quanta are specified in the current schedule table entry. When a task is rescheduled and at the end of each quantum, the length of the next quantum is stored in XTSLTS and XTSCTI.

If a task is interrupted, for any reason, before the end of its allotted time, the value of the cpu timer at the time of the interruption is recorded in PSATSA and XTSCTI, to account for the remaining time before the original task is to be time-sliced.

When the task is redispatched to complete its time slice, the remaining time available to that task (previously recorded in XTSCTI) is moved back into the cpu timer (resetting the clock). The cpu timer is again decremented by the hardware until that task's remaining time is used up, and the task is time-slice-ended by the system.

Whenever an interruption occurs during task execution, the task's timers, XTSUTI and XTSATI, are updated. Any remaining time (in XTSCTI), allotted to the task's time slice, is subtracted from its original allotment (in XTSLTS) to determine the elapsed time used by the task before the interruption. This elapsed time is added to the task's accumulated CPU-time cell (XTSATI) and, if a task time interruption has been established for the task, the elapsed time is subtracted from the XTSUTI field; see "Setting the Interval Timer."

Setting the Interval Timer

If no task time intervals have been established by programmers to generate interruptions at specific times, the time indicated in the cpu timer, when each new dispatch occurs, is the time from a task's XTSCTI field (the time remaining in that task's time slice). If, however, task time intervals have been established, whenever an interruption occurs in

the system; the cpu timer field is reset with the shortest period of time after which the system is to generate another interruption. Thus, the cpu timer is reset with the shorter interval contained in XTSCTI (time remaining in the time slice of the next task to be dispatched), or XTSUTI (programmed task-time interval after which an interruption is to be generated).

If the shorter time interval was found in XTSUTI (indicating that a task time interval will expire before normal time-slice-end), that task time interval replaces that task's time slice in XTSCTI. The task's accumulated cpu time cell (XTSATI) is incremented by the time used this time-slice (XTSLTS-XTSCTI) and XTSUTI is decremented by this value. The new XTSUTI is placed in XTSLTS and XTSCTI. The task is then redispatched until the new time in the cpu timer expires. When the task is rescheduled after the interruption from the task time interval, the schedule table time-slice allotment is again used to initialize XTSCTI and XTSLTS; this reestablishes a normal time-slice interval for the task.

REALTIME MAINTENANCE

Realtime for the entire system is maintained in the TOD (Time of Day) clock. Realtime intervals can be set by programmers. After a programmer-defined interval elapses, a system-generated interruption occurs to inform the programmer that the time has elapsed. Realtime interrupt values are maintained in the RTITIME field in the real-time interruption-pending queue in the resident supervisor (DSECT CHARTI).

RTITIME is a doubleword in the realtime interruption-pending queue that is maintained in the resident supervisor. Realtime intervals can be established in RTITIME by issuing the SETTIMER or SETTR macro instruction. Another user macro instruction, STIMER, will indicate an interval in one of a task's eight realtime software clocks; these intervals are eventually transmitted to RTITIME from these clocks, when the system issues SETTR.

TASK TIME MAINTENANCE

Task time statistics are maintained in each task's extended task status index (XTSI), which includes these time cells: XTSCTI, XTSLTS, XTSATI, and XTSUTI.

XTSCTI is a doubleword that indicates the length of time in a task's time slice when the task is initially dispatched. If the task is interrupted before time-slice end, this time cell contains an indication of time remaining in the task's time slice. The time is maintained as microseconds

XTSLTS is a doubleword that contains a value representing a task's total remaining time-slice allotment whenever it is rescheduled. The value remains constant while the task is executing; it is reset whenever the task is re-scheduled. This time is maintained as microseconds.

XTSATI is a doubleword that contains a value representing accumulated task CPU time, up to the time of the task's last interruption. It is calculated as the running sum of XTSLTS minus XTSCTI, then added to what's already in XTSATI. XTSATI is maintained in microseconds.

XTSUTI is a doubleword that contains a value representing a task-time interval after which a task's execution is to be interrupted.

This field is set by programmers to monitor execution of their programs. The time is maintained in microseconds.

TIMEKEEPING MACRO INSTRUCTIONS

System-defined macro instructions can be used to set and read time cells. Some of these macro instructions (discussed below) are available to the user programmer; others only to the system programmer. STIMER, TTIMER, EBCDTIME, SIR, and STEC are meant for use by the user programmer; they are described in Assembler User Macro Instructions; the system programmer's macro instructions are described in Part 2.

Both task time and realtime intervals can be established by using system-supplied macro instructions. When these programmed time intervals elapse, the system generates interruptions that tell programmers the time expired.

User programmers can issue the STIMER macro instruction to establish as many as eight concurrent realtime intervals and eight concurrent task time intervals, all of which are recorded in a table in the task monitor (the table has 16 software clocks for each user). As each interval elapses, control is returned to the user's program. Although STIMER sets these 16 software clocks, it is the inner system macro instructions that place these clock values in the appropriate system time cells. Thus, the system macro instruction SETTU places task intervals into XTSUTI; SETTR (from virtual storage) and SETTIMER (from resident storage) place realtime intervals into RTITIME in the realtime interval table (DSECT CHARTI) in the resident supervisor. User programmers can also use the SIR and STEC macro instructions to establish these task time or realtime intervals.

The user macro instruction TTIMER can be used to test any of the 16 software-timer intervals previously set by a programmer. Data returned to the programmer indicates either the time remaining in the interval timer he is testing, or that the time has expired.

The TSEND system macro instruction, which causes a task to be placed in the delay state, sets a realtime interval, after which the task will be redispatched if no other interruption occurs. This realtime interval is equal to a system-defined value, known as "delay time," recorded in the system table field SYSDLY. The system issues SETTIMER to place this interval in RTITIME; the CANCL system macro instruction can be used to cancel an interval established by SETTIMER.

Reading System Time

The realtime setting that indicates the current instant in microseconds can be determined by issuing the REDTIM system macro instruction or the EBCDTIME user macro instruction.

REDTIM causes the TOD clock to be converted to microseconds and added to SYSYMD and SYSTOD; it causes the resulting double-precision fixed-point number to be returned in registers 0 and 1. This is the number of elapsed microseconds since March 1, 1900.

EBCDTIME allows users to specify the format in which they want the realtime returned to them as: years, months, days, hours, minutes, seconds, or tenths or hundredths of seconds.

The system macro instruction XTRXTS extracts the values set in the time cells:

XTSUTI (explained above under "Task Time Maintenance.")
XTSATI (explained above under "Task Time Maintenance.")

XTSETI (a field indicating the estimated run time for a task. It must be set using the SETXTS system macro instruction.)

Another system macro instruction, XTRTM, extracts a task's accumulated CPU time, in milliseconds.

GENERALIZED TRACE FACILITY

The Generalized Trace Facility (GTF) allows hardware and software maintenance personnel to dynamically record hardware and software events.

Using GTF, personnel can record, for later analysis, all interrupts from a particular device or group of devices, the SVCs and/or program checks caused by a specific task or all tasks, or entries into RTAMs device modules for a particular device or group of devices or network addresses.

If the interrupt rate is too high for ordinary recording, or if only the interrupts for a particular event are wanted, The user can dynamically set up a circular log in the supervisor and after the event has occurred, have the log returned to the task and displayed.

On a high interrupt rate device like the 2305 drum, or attempting to trace all disk I/O with 100 tasks running, no attempt should be made to write all the trace records to a data set, because not only will this degrade performance considerably, but it may cause a system failure due to insufficient storage; although GTF will automatically turn off a trace if there is insufficient space for a recording block, it is possible to satisfy GTF requests and then have a condition wherein there is not enough space to satsify the next system request.

Once a trace is started, it can be ended in one of four ways:

1. the user issues a command to end the trace.
2. the task being traced abends.
3. the trace owner's task abends or logs off.
4. not enough storage exists to allocate a trace block.

OPERATION OF THE GTF FACILITY

The system programmer or maintenance person enters the GTF command specifying the wanted options and a data set name to store the trace records. There is a limit of 255 concurrent GTF reqeusts for the system and only one GTF can be active against a resource at any one time. Two or more GTF requests against the same SDA, for I/O, are not allowed.

The GTF module, CZNCA, validates the options, defines and opens the output data set, if one has been requested, SIRs for the trace external interrupt, and issues the trace SVC to the supervisor to activate the trace.

The supervisor GTF module, CEDTRACE, validates the trace request input and asssigns a trace ID for the request. CEDTRACE sets the trace flags in the control blocks associated with the resource to be traced, allocates and builds the trace blocks and returns a positive response to the task.

As each trace block is filled, the supervisor will queue the block on the task as an external interrupt using the interrupt code specified in

the trace request. The GTF external interrupt routine will receive control and writes the block into the output data set. Each block from the supervisor has a trace ID and a sequence number in it. There are also flags set in the header specifying the trace status and if ended, the reason for ending. The GTF moduele does not consider the tracae ended until receipt of the stopped flag from the supervisor.

GTF BLOCK AND RECORD FORMAT

The GTF block built by the supervisor is 2048 bytes in length and contains a 32 byte header. The first 16 bytes are used as the MCB header when the block is shipped to the task. The next 16 bytes are the trace block header and contain pointers to the first, next and last entry position within the block. Following the pointers is one byte of flags, a one byte trace ID and a halfword block sequence number.

Each GTF entry within a block is a fixed length of 48 bytes. The first 16 bytes of an entry are a header supplied by the supervisor trace module. The header contains a time stamp, CPU address, record length, AID and FID bytes, and an event ID. The remaining 32 bytes are trace dependent.

For a more detailed description of the trace block, see DSECT CHATRAC

The assigned event IDs are as follows:

System Wide Event IDS

X'0018' - external interrupt event
X'0020' - SVC interrupt event
X'0028' - program check interrupt event
X'0038' - I/O interrupt event
X'019C' - SIO event
X'019D' - TIO event
X'019E' - HIO event

RTAM event IDs

X'8001' - PIU read completion
X'8002' - PIU write completion
X'8003' - RTAM module entry

Task event IDs

X'4000' - task RTAM interrupt
X'4018' - task external interrupt
X'4028' - task extended program interrupt
X'4038' - task I/O interrupt entry

TIME CONVERSION ROUTINE

A number of privileged conversion routines are provided to enable you to convert time data, in any of several formats, into a form you can use with the macro instructions SETTR and SETTU. Two types of conversions are performed: type-T, used for operations with the SETTU macro instruction, yields a 32-bit binary time interval in microseconds; type-R, used for operations with the SETTR macro instruction, yields a 64-bit binary time interval in microseconds elapsed since March 1, 1900 (see "Timekeeping"). Two different forms of input data may be used for type-T conversion (0 and 1); six forms (0-5) may be used for type-R. Figure 10 summarizes the different input forms.

Input Data Code	Input Form
0	Time interval in hours (h), minutes (m), seconds (s), tenths (t), and hundredths (h) of seconds; eight BCD characters: hhmmsstth
1	Time interval in milliseconds; 32-bit binary number
2	Time of day in hours (h), minutes (m), seconds (s), tenths (t), and hundredths (h) of seconds; eight BCD characters: hhmmsstth
3	Day of week; four left-justified BCD characters: MOND, TUES, WEDN, THUR, FRID, SATU, SUND
4	Day of month; two left-justified BCD characters: 00 through 31
5	Day of year; eight left-justified packed decimal characters: 00yyddd+

Figure 10. Input formats accepted by the time conversion routine

To use the time conversion routine, you must put a pointer to a parameter list in register 1, the return address in register 14, and the address of the time conversion routine in register 15. It looks like this:

```
         LA       1,PARAM          POINTER TO PARAMETER LIST IN
*                                  REGISTER 1
         L        15,=V(CZCJXA)    ADDRESS OF CONVERSION ROUTINE
         BASR     14,15            GO THERE
         RETURN
PARAM    DC       C'1'             FORM OF INPUT DATA - 0,1,2,3,4,
*                                  OR 5
         DC       C'T'             TYPE OF CONVERSION - T OR R
         DC       H'0'             NOT USED
         DC       D'DATA'          INPUT DATA PLACED HERE - RESULTS
*                                  FOUND HERE
```

After completing the requested conversion, the time conversion routine returns control to the address found in register 14. The results are placed, right-aligned, in the second and third words of the parameter list.

Note: The SETTU macro instruction expects a time value in milliseconds; if you use the time conversion routine to get a time interval (type-T), you must divide the result by 1000 to convert it to milliseconds.

Figure 11 lists the meaning of the results obtained from the various conversions.

Conversion	Result
T0	Time interval in microseconds
T1	Time interval in microseconds
R0	Current time + input time interval in microseconds from March 1, 1900
R1	Current time + input time interval in microseconds from March 1, 1900
R2	Next occurrence of input time in microseconds from March 1, 1900
R3	Next occurrence of day of week in microseconds from March 1, 1900
R4	Next occurrence of day of month in microseconds from March 1, 1900
R5	Next occurrence of day of year in microseconds from March 1, 1900

Figure 11. Results of time conversion

EVALUATING SYSTEM STATISTICAL RECORDING FIELDS

The internal relationships that characterize the operation of TSS system programs and user load are difficult to evaluate. To help in an evaluation of these relationships, data pertaining to the system and individual tasks are dynamically maintained by TSS. These data include, among others, paging counts, real memory utilization, software queue processor times, CPU utilization, and scheduling counters. These data indicate, among other things, the CPU workload, I/O load balancing, schedule table efficiency, and individual task loads.

Two classes of statistics are collected; gloval system wide information and local task oriented data. These statistics are recorded as the system is running in fields located in various system tables and modules. Many of the statistics are recorded in the system status table (CHBSST) in main storage. Additional statistics are dynamically maintained in the fields indicated in three dsects; CHAPXS, CHASTV, and CHATSX.

Analysis of System Status Statistics

The statistics, maintained dynamically by the system, are available to the system programmer through use of the statistics sampling macro instruction SAMPLE. Execution of this macro instruction causes statistical data to be collected and recorded in the virtual storage page in

which SAMPLE is issued. System programmers can then write their own data analysis programs to evaluate the data. Proper analysis allow system managers and programmers to evaluate the workloads being processed at their installation in relation to the TSS environment.

Once statistics have been analyzed, the system programmer can reinitialize statistical data fields in the system to zero by issuing the ZEROSST macro instruction. Subsequent dynamic recording of statistics will reflect the system's performance since the time at which ZEROSST was issued rather than since Startup.

ADJUSTING ASSEMBLER CONSTANTS

The TSS Assembler normally secures working space for internal use and for producing output information. Occasionally, source programs cause an overflow of one or more of these work areas. This occurs only when an exceptionally large program (for example, an application programmer's own language-processing program) is being assembled. The symbols associated with six of these work areas have been defined as entry points to allow O and P authority programmers to alter (via use of VSS) the size of these work areas when such a program is assembled. Thus, the fullword constants at these entry points may be altered prior to an assembly to increase sizes of the work areas for which the assembler will issue GETMAIN macro instructions.

The constants must not be altered once an assembly has started; in such cases, the assembler's virtual storage allocation routine will issue a completion code 1 ABEND when it detects an attempt to free more work area than was assigned by GETMAIN. The fullword constants, the work areas to which they apply, and several possible causes for their being overflowed are indicated in Figure 12.

CAUTION: Since these constants reside in public, read-only code, they must not be altered if any other user is assembling.

ALTERING CONSTANTS

When an assembly is terminated because of work area overflows, the diagnostic message issued by the assembler names the work area that overflowed. The authorized programmer can then use a SET command to alter the constant associated with the named work area. For example, if a diagnostic message indicates the PMD work area overflowed, the programmer might issue:

```
SET     CEVPMD.(,4)=4
```

This would increase from two to four the number of pages that would be added to the number of text pages divided by eight.

ESTIMATING WORK AREA STORAGE REQUIREMENTS

A knowledge of PMD and ISD control block layouts and the sizes of entries created for external references (REFs), address constants, etc., may be helpful when analyzing PMD or ISD work area overflows. System programmers can estimate the number of additional pages based on the entry size multiplied by the number of additional entries required. The DSECTs CHATDY and CHAISD, describe components for the PMD and ISD and may aid in determining the storage requirements.

Constant	Work Area	Normal Size	Reasons for Overflow/Comments
CEVW1	Work 1	100 pages	Too many USING or DROP statements
			Insufficient room for 2-word cross reference items
			Nesting of macro calls causes generation of macro level dictionaries that required too much space
CEVW2	Work 2	255 pages	Too many symbols in name fields
			Too many source statements
			Too many macro-generated statements
			Too many continued lines
			Insufficient work space for building control section dictionary
CEVW3	Work 3	20 pages per increment, more available as needed to hold source statements	Insufficient virtual storage available for any request; increasing size of 20 will normally not help. Unload all modules possible; reassemble
			Assembler's VMTABLE filled up. Increase size from 20 pages per increment.
CEVXL	External Name (EXT NAM) List Area	2 pages	Number of control sections and ENTRY operands more than 1022
CEVPMD	PMD Work Area (residual count)	2 pages plus (number of text pages divided by 8)	Too many ENTRY or EXTRN operands with textless or low text module (TEXT=instructions or constants)
CEVISD	ISD Work Area	0 pages plus number of pages in Work 2	The value of CEVW2 too small to contain ISD
			Too much USING/DROP information and symbolic name information for work area

Figure 12. Assembler constants, changeable for large assemblies

RELEASING INTERLOCKS AT ABEND

Many interlocks in shared tables (coded in TSS system programs) are released automatically by system routines (for example, the RESTBL header) if an abnormal end (ABEND) occurs. However, system programmers may set locks on shared tables that are not automatically released when their task is abnormally terminated. In such cases, other tasks attempting to use these tables are prevented from doing so because of

interlocks left on the shared table by the abnormally ended task. To eliminate this problem, system programmers must establish entries in the ABEND interlock release (AIR) table for all interlocks they set that have no predefined system release procedure. These entries contain the V- and R-type address constants to a routine (that must also be coded by the system programmer) which is to release the interlock in case of an ABEND. For interlocks of this type, AIR table entries must be created before locking the interlock byte. A suggested procedure for creating an AIR table entry is indicated below:

```
IRSPSECT PSECT
LOCKSW   DC      X'00'                 INTERLOCK SWITCH BEING USED BY
                                       SYSTEM PROGRAMMER TO LOCK
                                       SHARED TABLE:
                                          00 - UNLOCKED
                                          FF - LOCKED
          .
          .
LOCKLOG  LOGVLOCK
          .
          .
SW       DC      X'00'                 AIR TABLE ENTRY SWITCH
                                          00 - NO AIR ENTRY EXISTS
                                          FF - AIR ENTRY EXISTS
          .
AIRLST   CALL,(IRSPSECT,IRSRTNE,USEDATA,),MF=L
+        DC      V(IRSPSECT)
+        DC      R(IRSRTNE)
+        DC      A(USEDATA)            POINTER TO USER DATA
          .
          .
USEDATA  DS       D                   CZACS SCRATCHPAD
          .
          .
AIRSET   CSECT                         SYSTEM PROGRAMMER's ROUTINE IN
                                       WHICH INTERLOCK (for example,
                                       LOCKSW) IS TO BE SET ON
          .
         CLI     SW,X'00'              HAS CZCACS1 BEEN CALLED
         BNE     SKIP                  YES, AIR ENTRY ALREADY EXISTS
         CALL    CZACS1,MF=(E,AIRLST)  NO, CREATE AIR ENTRY (i.e.,
                                       CALL ROUTINE THAT CHAINS ENTRY
                                       FROM AIRSET's PSECT INTO AIR
                                       TABLE)
         MVI     SW,X'FF'              SET AIR SWITCH TO INDICATE
                                       ENTRY EXISTS
          .
SKIP     SETVLOCK LOCKSW,LOCKLOG       ONCE HAVING SET AIR ENTRY, SET
                                       INTERLOCK ON!
          .                            NOW IF YOUR TASK IS ABENDED,
          .                            LOCKSW WILL BE AUTOMATICALLY
          .                            RELEASED BY THE ROUTINE YOU
          .                            SPECIFIED IN AIRLST
         OPNVLOCK LOCKLOG
          .
         CALL     CZACS2,MF=(E,AIRLST)
          .
         MVI      SW,X'00'
```

The system programmer would also include the following steps in the interlock release routine:

```
IRSRTNE  ENTRY
          .
```

```
CLRVLOCK LOCKLOG              CHECK AND RESET LOCKSW BYTE
  .
  .
MVI      SW,X'00'             TURN AIR SWITCH OFF SO CZACS1
  .                           WILL BE CALLED AGAIN IF ROU-
  .                           TINE IS ENTERED AT SUBSEQUENT
  .                           ABEND
RETURN                        RETURN TO CALLER
```

The AIR entry is automatically released by the system when ABEND dispatches the system programmer's interlock release routine.

PUBLIC POOLS -- GENERAL DESCRIPTION

The purpose of public pools in TSS is to divide public storage volumes and uses into logical groups. These logical groups, or pools, consist of one or more volumes with consecutive relative volume numbers named by an eight-character poolid. One pool, called the system pool, is required at system startup time. All other pools are dynamically added and deleted from a running system.

Each pool is a self-contained entity. All datasets necessary for a pool are contained within the pool. Users are joined to a specific pool, rather than to a TSS system, and may own datasets only within that pool. Datasets may, however, be shared across pools. A special user in each pool, called the pre-joined user owns all the pool's system datasets such as: SYSCAT, SYSUSE, and SYSBWQ. The pre-joined user also has the responsibility for joining and quitting users to his/her pool and for the maintenance of the pool. TSS***** is considered the pre-joined user for the system pool.

The structured public storage allows the separation of users and their data into reasonably-sized, independent groups, divorced from any particular TSS system. Groups of users may be moved from one TSS system to another. New releases of TSS and PTFs affect only the system pool.

Public storage backup and maintenance are performed on a pool rather than the entire system. Catastrophic DASD failure is localized to one pool. Since each pool contains a SYSCAT, catalog paging bottlenecks are reduced. Special purpose pools for benchmarks and system testing may be created.

The volume label of each volume in a pool contains the eight-character poolid and a flag to indicate that it is part of either a system or a public pool. The volume label of the first volume in a pool also conatins the pre-joined userid and pointers to SYSCAT, SYSSVCT, and SYSVOL datasets. SYSCAT is the system catalog for the pool; SYSSVCT is the index of user catalogs for the pool; and SYSVOL is a list of all volumes that make up the pool. Each pool also contains SYSUSE, SYSBWQ, and SYSPLIB datasets. These datasets are owned by the pre-joined user and are normally contained on the first volume of a pool. SYSPLIB is a pool oriented VP dataset that is data deffed between USELIB and SYSLIB. It is used to contain object modules related to a particular pool.

Several virtual memory tables are used to control the active pools on a system. These tables are updated during ADDPOOL and DELPOOL processing. The active pool index (CHBAPI) contains the status and control block pointers for all pools currently known to the system. The pool volume table (CHBPVT) contains a list of volumes for each pool and is used to control space allocation for datasets. The userid table is used to determine the poolid for a specific userid. Entries in the virtual batch work queue are added from a pool's SYSBWQ dataset during ADDPOOL processing and deleted during DELPOOL processing.

CONVERTING TO A POOL SYSTEM

To convert an existing TSS system into a single pool system, the volume labels of all public volumes must be updated to conatin the new pool related information. Also, the SYSVOL dataset must be built on the first volume. This is a 'one-time' process and is performed by the CNVTPOOL command. The modifications to the volume labels and the SYSVOL dataset do not affect the operation of pre-pool systems.

ADDING AND DELETING POOLS

The startup process automatically adds the system pool; all other pools must be added by the system operator. The shutdown process deletes all active pools including the system pool; pools other than the system pool may be deleted by the system operator. The commands ADDPOOL and DELPOOL provide this facility.

The ADDPOOL command provides a means by which a pool can be added to the system, or put into a maintenance status. Maintenance status restricts all users in the pool from logging on except the pre-joined user. This enables the pre-joined user to perform maintenance and backup procedures without interference from other users. A pool may be removed from maintenance status by issuing another ADDPOOL command (which adds the pool to the system) or by issuing a DELPOOL command (which deletes the pool from the system).

The DELPOOL command will optionally force active users off a pool. If users are not forced, the pool is marked in 'delete status'. This delete status prevents new users in the pool from logging on, and prevents shared datasets from being opened by users in another pool. A pool may be removed from delete status by issuing an ADDPOOL command. To delete a pool already in delete status, another DELPOOL command must be issued.

BUILDING A NEW POOL

A new one-volume pool is built from an empty private volume by the BLDPOOL command using the MODE=NEW parameter. With userid TSS***** the BLDPOOL command builds the SYSVOL dataset and empty datasets for SYSPLIB, SYSBMQ, and SYSCAT. The datasets SYSUSE and SYSSVCT are built and include entries for th pre-joined user. Also, a USERCAT for the prejoined user containing catalog entries for the above datasets is constructed. Finally, the volume label is modified to contain the pool information. Once built, a new pool may be added to a system with the ADDPOOL command. Users may then be joined to the pool by logging on the pre-joined user and issuing the JOIN command.

ADDING VOLUMES TO EXISTING POOLS

A new volume may be added to an existing pool by the BLDPOOL command using the MODE=ADD parameter with the pre-joined userid. The BLDPOOL command updates the pool's SYSVOL dataset to include the new volume and modifies the volume label of the new volume to contain the pool information. The new volume will become available for storage allocation the next time the pool is deleted and then added to the system. After the new volume has been deleted-added to the system, storage allocation will occur on this volume after the next shutdown-startup sequence.

PARTITIONING AN EXISTING POOL

The process of partitioning an existing pool into two separate independent pools involves several steps. First, the breakpoint (the relative volume number of the first volume of the new pool) must be determined. Also, it must be determined that the available disk storage of each of the two pools must be large enough to contain all the datasets owned by the users that will eventually belong to each pool.

Second, once the breakpoint is determined, external storage allocation for each user of the existing pool is restricted to one of the two new pools by the SETRVN command; this establishes the pool to which each user will eventually belong. Allocation for new datasets and expansion of old datasets will only be made within the volume range specified by the SETRVN command.

Third, to ensure that users do not own datasets outside the volume range specified in the SETRVN command, a MOVEUSER command must be issued for each user in the existing pool. MOVEUSER will check each dataset including USERCAT for a user and copy the dataset if any part of it is outside the volume range. The sharing information and the time and date stamp are maintained. The volume range and MOVEUSER status information are kept in the SYSSVCT dataset.

Fourth, the BLDPOOL command with a MODE=PART parameter is issued by the pre-joined user. This command will test each user in the existing pool for relative volume number range consistency and MOVEUSER status. This command is canceled if the relative volume range of any user overlaps the breakpoint, or if any user has not been checked with a MOVEUSER command.

The SYSVOL dataset for the new pool is built, and the SYSVOL dataset of the existing pool is updated. Empty datasets for SYSPLIB, SYSDWQ, and SYSCAT are constructed for the new pool. The datasets SYSUSE and SYSSVCT are built and include the new pool's pre-joined user. A USERCAT for the new pre-joined user is built to contain catalog entries for the above datasets.

The existing pool's SYSSVCT is read to determine which users are to be included in the new pool and both the existing and new SYSSVCT and SYSUSE datasets are updated accordingly.

Finally, the volume labels of the volumes in the new pool are updated. The new pool may now be added to the system by first deleting the existing pool and then adding the two 'new pools'.

MAINTENANCE

Public storage maintenance is now performed on a pool basis rather than on a system basis. Since pools, including the system pool, are independent of each other, dump-restore of all volumes in a particular pool ensures dataset/catalog consistency. Commands such as PATFIX operate on a single pool at a time. A new command, DSCBS, replaces the CPS, LPDS, CVV, and RPS commands, and operates on a pool basis. The command BLDSVCT is provided to rebuild SYSSVCT for a pool in maintenance status. BLDSVCT replaces the SYNCCAT command. A USERCAT may be rebuilt by the new FIXCAT command.

DATASET DSCB RECOVERY

This item will provide for dscb error analysis and recovery.

In order to accomplish this 2 new types of DSCB's have been defined, a type 2 format "E", and a new format "G". The type 2 "E" contains the same dscb type code X'01' but is marked as a type 2 by a bit in the dseflg field of the dscb. The new "G" dscb contains the code X'03' in the dscb type field.

The EPEs (external page entries) for both new dscbs have also been redefined. The new format is:

```
srvxpppp,   s    = slot 0-f
            rv   = relative volume number 0-ff
            x    = zero
            pppp = relative page number 0-ffff
```

In addition, the following new fields have been added to the "G" dscb to aid in dscb integrity and recoverablity:

```
DSGSEQ(4)   - dscb sequence counter 1,2,3,,,n
DSGFMTE(4)  - pointer to the format "E" dscb
DSGCTEPE(1) - count of epes in the dscb
DSGMAX(1)   - maximum number of epes this dscb may contain
```

The format "E" pointer contained in the format "G" dscb is used as an anchor to the format "E". The sequence counter is used to indicate where in the chain a dscb slot belongs. A new command "FIXDSCB" (see commands section) has been written to analyze dscb slots and their contents, and attempt recovery if errors are detected. This command can only analyze dscb chains in the new format.

CZCEW (write dscb) has been rewritten to create only new format dscbs. However, as released CZCEW will link to CZCEV (the old write dscb routine) and continue to create dscbs in the old format.

Creation of new or old dscbs is controlled by the following patches:

```
SET CZCEWALL = X'4700' allow new dscb creation
             = X'47f0' create old format dscbs for all datasets

             the above patch will cause new format dscbs to be
             converted back to old format dscbs.

SET CZCEWNEW = X'4700' create new dscbs for all datasets
             = X'4770' create new dscbs for new datasets only.
```

VALIDATING DSCB SLOTS

System programmers occasionally find it necessary to modify or validate DSCB slots. The checksum is used to validate DSCB slots.

If a programmer reads in (via a SETXP system macro instruction) a page of DSCBs and locates the DSCB he wants to examine, he can verify that the DSCB was read in correctly by executing the checksum procedure described below. He computes the checksum of that DSCB and compares it with the checksum value already recorded in bytes 255 and 256 (the checksum field) of that DSCB. When the checksums match, the DSCB is assumed valid. If they do not match, that DSCB slot is assumed to be erroneous. The system programmer should then attempt to recover, as far as possible, from the checksum error. The system service routine, DSCBREC (see System Service Routines), can be used to attempt this recovery. The DSCB page can be written on external storage through use of the PGOUT system macro instruction.

Whenever a system programmer modifies a DSCB, he must recompute the checksum value, record it in the DSCB checksum field, and rewrite the DSCB page to external storage.

Checksum Procedure

The following standard procedure is used for computing the checksum values for DSCB slots. The first 63 words of the DSCB are summed and the sum complemented. The high-order half (bits 0-15) of the result is then added to the low-order half (bits 16-31) and the low-order half of the result is placed in the last halfword (bytes 255 and 256) of the DSCB.

This sample code illustrates the checksum procedure. Upon entry, register 8 contains the DSCB slot number, and CKAD contains the virtual storage address of the DSCB work page.

```
         SLL     R8,8                SLOT NO. *256
         AL      R8,CKAD             + ADDR. OF DSCB PAGE.
         LA      R14,244             SET COUNT AND
         L       R15,248(R8)         LOAD LAST WORD
CKSUM    AL      R15,0(R14,R8)       ADD IN PREVIOUS WORD AND
         S       R14,CONCK           REDUCE WORD COUNTER BY ONE.
         BC      11,CKSUM            IF NOT FINISHED, CONTINUE.
         LCR     R15,R15             COMPLEMENT SUM AND
```

```
          STH     R15,CKSX              STORE LOW-ORDER HALF.
          SRL     R15,16                SHIFT DOWN HI-ORDER HALF,
          AL      R15,CKS               ADD IN LOW-ORDER HALF AND
                                        PERFORM STH OR CH.
           .
           .
           .
        ***IF DSCB WAS MODIFIED, STORE NEW CHECKSUM VALUE***
          STH     R15,254(R8)           STORE RESULT IN DSCB.
           .
           .
           .
        ***IF SIMPLY VERIFYING DSCB WAS READ-IN CORRECTLY***
          CH      R15,254(R8)           COMPARE WITH EXITING CHECKSUM
           .
           .
           .
CONCK     DC      F'4'                  COUNT DECREMENT CONSTANT
CKS       DC      H'0'                  SUMMATION WORD
CKSX      DS      H
CKAD      DS      F                     DSCB WORK PAGE ADDRESS
```

VIRTUAL MEMORY SUPERVISOR CALL INSTRUCTIONS

Virtual Memory supervisor calls are those SVCs whose processing programs are in virtual storage; these SVCs use operand codes 0 through 127. Codes 0 through 99 are reserved for nonprivileged program defined services while codes 100 through 127 are reserved for privileged program defined services.

Many of these SVCs can be executed only from nonprivileged code; if a privileged module attempts to execute them, diagnostic messages will be issued. When a nonprivileged supervisor call is issued, the supervisor passes it back to the task monitor as a task-SVC; no task program interruptions are generated. The task monitor transfers control to the appropriate privileged (or nonprivileged) program for processing.

Nonprivileged programs can neither read, write, nor transfer control to privileged programs directly; some form of interruption (for example, the interruption caused by execution of an SVC instruction) is required.

The Virtual Memory SVCs described in this publication are listed in Appendix B.

REAL MEMORY SUPERVISOR CALL INSTRUCTIONS

The SVC queue processor controls the execution of SVCs 128 through 255. Codes 128 through 143 are reserved for installation use, codes 144 through 169 are reserved for TSSS, and codes 170 through 255 are resident supervisor SVCs.

System programmers (P or O) may issue all resident supervisor SVCs (170-255). Any program operating in the privileged-program state (VPSW p-bit = 0) -- even if being run by a user-programmer -- may issue all the Real Memory SVCs. The Real Memory SVCs described in this publication are listed in Appendix B.

If a nonprivileged program being run by a user-programmer attempts to issue a Real Memory supervisor call, the resident supervisor may create an extended program interrupts. When the task monitor receives the interruption, it calls DIAGNO. Generally, supervisor calls that can disrupt the operation of TSS are privileged. Supervisor calls that allow access to private information are also privileged.

Usually, the operation requested by a Real Memory SVC is a synchronous one which is completed by the resident supervisor before it returns control to the task that issued the SVC. The principal exception to this is IOCAL, an asynchronous SVC, which is processed concurrently with the issuing task.

Task program interruptions, which may result from improper use of these macro instructions, can be found in Appendix C.

ACCESSING SYSTEM DATA SETS

Access to the system catalog and the user table is restricted to system programmers having an authority code of 0 and to certain privileged routines, such as catalog service routines. Access to all other system data sets is available to system programmers having either authority code 0 or P.

SYSER DUMP

To facilitate your monitoring of the system, dumps can be taken to show real or virtual storage as they existed at the time an error was detected. When the system error processor is called by the ERROR macro instruction (SVC 254) or the SYSER macro instruction (SVC 228), a message is displayed at the operator's terminal to record the error. The system then enters the wait state and the operator uses the support system to take the dump and, if hard copy is desired, to print it. After the dump has been taken and control returned to the system, processing continues as described under the ERROR and SYSER macro instructions.

If the call to the system error processor came from main storage (real core) via the ERROR macro instruction, the following message is displayed at the operator's terminal:

```
<ER> RM mmnn{MIN|MAJ}userid TID{C|N}{SYSIN}hhmm
CPU# SP=SVC PSW
MODULE: module name  base address
REGISTERS: 0 - 15
```

where the elements have the following meanings:

- <ER> identifies the message as being issued by the system error processor.
- RM identifies the call as having come from main storage (real core).
- mmnn is the four-digit ERROR code that identifies the call (see the description of the ERROR macro instruction for an analysis of the code).
- (MIN) identifies the error as type-1; (MAJ) identifies the error as type-2.
- UID=userid identifies the user whose task was running when the error was detected by means of his eight-character user identification, or userid, which is contained in the TSI.
- TID=taskid identifies the task that was running when the error was detected by means of its four-hexadecimal-digit task identification, or taskid, which is contained in the TSI.
- C indicates that the task that was running when the error was detected was conversational; N that it was nonconversational.

- SYSIN=xxxx specifies the four-hexadecimal-digit symbolic device address contained in the TSI.
- hhmm indicates the time in hours (hh) and minutes (mm) at which the error was detected.
- CPU= indicates the number of the CPU
- SP=SVC PSW at the time of the SYSER call
- MODULE: identifies the module that issued the SYSER: call and its base address
- REGISTERS: indicates the contents of general registers 0 - 15 at the time of the SYSER SVC

If the call to the system error processor came from a privileged routine in virtual storage via the SYSER macro instruction, the message is modified slightly and takes the form:

```
<ER>VM aabbcccnn{MIN|MAJ} userid TID{C|N}{SYSIN}hhmm
CPU# SP=SVC PSW
MODULE: module name  base address
REGISTERS: 0 - 15
```

where those elements in common with the main storage (RC) message have the same meaning and the remaining elements have the following meanings:

- VM identifies the call as having come from virtual storage.
- aabbcccnn is the nine-digit SYSER code that identifies the call (see the description of the SYSER macro instruction for an analysis of the code).

The information identifying the task (UID, TID, SYSIN) is always valid when the call is from virtual storage; it is not necessarily valid when the call comes from main storage. The task identified in the message resulting from a main storage call is the last task dispatched by the supervisor. However, since the supervisor can perform many functions before dispatching another task, the task names in the message may not be the one causing the problem. Whether or not the task information is valid must be determined by the system programmer.

For conventional tasks, the symbolic device address is that of the terminal from which the input stream is being entered, while for nonconversational tasks, it is the volume on which the SYSIN data set resides.

For all errors detected in bulk I/O and batch monitor tasks as well as in the main operator's task, the user ID is SYSOPER0. To distinguish among these, you must use the four-hexadecimal-digit task identification.

RELIABILITY AIDS

AUTOMATIC ISA REPLACEMENT

STARTUP saves the critical portion of the ISA in the supervisor and the pointer and length of the saved ISA in CHBSYS. CEAA2 and CEAJI compare the saved PSWs with the task's ISA PSWs. If they are not the same, the ISA rebuilder, CEAA22, is called. This permits a task whose

ISA has been overwritten to have the critical portions of its ISA restored, thus allowing the task to be deleted gracefully; otherwise, the task would go into an unending loop and could not easily be forced from the system.

Barrier Pages

When STARTUP loads the virtual memory modules it generates a barrier page for each LLIST macro in CHBVM with operands BARRIER=PRIVATE/SHARED, putting the pages in private or shared memory as requested. STARTUP will also generate a barrier page immediately following the ISA plus one at location 'FFF000'.

READ ONLY PAGE OVERWRITE PROTECTION

STARTUP identifies all csects with the read only attribute, and sets a flag in the External Page Table entry for each read only CSECT. This can be overruled by a parameter ROPROT=N in the load list.

If a read only page is found to be changed by the TSS supervisor a '8388' real core minor SYSER will be issued. Register 0 will contain the VMA and register 1 will contain the RMA of the changed page (for shared pages, register 0 is relative to the start of the shared page table). After issuing the SYSER the changed version of the page is thrown away and a fresh copy will be used the next time the page is referenced.

Two switches in CHBSYS may be used to alter the read only page protection. The switches are 'EF' bytes into CHBSYS. The first switch (X'80') will stop all read only page protection. The second switch (X'40') will detect changed read only pages, but will turn off the read only bit in the XPT and continue with the changed pages.

MODIFYING SYSTEM FACILITIES

To change TSS, you will probably follow a procedure like this:

1. **Define the function to be accomplished.**

2. **Identify the modules to be added, changed, or deleted.**

3. **Define the interface of these modules with all other TSS modules. The control section dictionaries of modules in the system provide you with a listing of all the module's external references (REFs) and external definitions (DEFs). This is a start in determining, for example, how an existing module fits into the system. Care must be exercised, as this information may be deceptive. For example, an external address can be loaded into a register, and the register (instead of the external address) can subsequently be referred to in the program. You might see this:**

```
BOLD      L       5,=V(CHBSYS)   SYSTEM TABLE ADDRESS
          USING   CHASYS,5       FORMAT OF SYSTEM TABLE
SNEAKY    L       6,SYSLOW       EXTERNAL REFERENCE
          L       7,60(5)        EXTERNAL REFERENCE
```

The symbol CHBSYS is an external symbol and would appear in the control section's dictionary as an external reference (REF). The reference to SYSLOW would not appear as an external reference, though, and the reference 60(5) isn't even a symbolic reference. The cross-reference dictionary would show you that statement BOLD refers to the externally defined symbol CHBSYS; you have to figure out that SNEAKY also refers to it.

Unfortunately, there's no convenient way to determine what programs refer to the external symbols defined in a given program. The instruction:

```
ENTRY ABCRJG
```

allows other programs to refer to the symbol ABCRJG. There is no guarantee, however, that other programs will actually refer to ABCRJG. Consequently, if you delete a program from TSS, you have no systematic way to determine which programs refer to the program you're deleting. You can discover the references only by carefully studying the function of the program being modified or replaced and by understanding its role in the overall design.

You might be tempted to list all the external symbol dictionaries of all the object modules that make up TSS, as a way of determining their interdependencies. This might be helpful, but it is not foolproof. Some programs set up registers with external addresses for use by other programs that know what the registers are supposed to contain. A program using registers set up by another program might not contain a single explicit external reference. You might see this:

```
OBVIOUS  L      6,=V(CHBSYS)    LOAD EXTERNAL SYMBOL
         L      15,=V(SNEAKY)   LOAD ADDR OF SUBROUTINE
         BASR   14,15           TRANSFER
```

The subroutine might look like this:

```
         USING  *,15            DECLARE BASE
SNEAKY   L      8,12(6)         HIDDEN EXTERNAL REFERENCE
```

The external reference to CHBSYS would never show up in the external dictionary of the program module containing SNEAKY. Note that the best coding practice would have been to cover register 6 with a DSECT for CHASYS and address the field symbolically.

4. Write the assembler statements.

5. Assemble and test the new or amended modules and store them in the same library.

6. Update the TSS system data sets using the procedures described in System Generation and Maintenance.

PROGRAM CONTROL SYSTEM (PCS)

The program control system (PCS) is not, in general, applicable to system programs. For example, one PCS command, the CALL command, always transfers control in the nonprivileged state and, therefore, cannot be used to transfer control to a privileged program. The privileged system programmer (authority code O) who wishes to directly invoke a privileged program must do so by including a BPKD macro instruction in the program and issuing a BUILTIN command. (See the BPKD macro instruction for a full explanation.)

Three PCS commands can be used with privileged system programs. Through the SET command, privileged system programmers can modify privileged, public control sections; this is the only way in which such control sections can be changed from a terminal. In addition, privileged public control sections may be examined by the use of the DISPLAY and DUMP commands. The following discussion covers the precautions you must observe.

Each D-class user is assigned an authority code by JOIN: code P specifies a system programmer, O specifies a privileged system programmer, and code U specifies an ordinary user. When someone logs on, this authority code is used to govern the operation of the dynamic loader and the use of PCS.

The dynamic loader ignores or overrides control section attributes depending on the programmer's authority code and the library from which the module is loaded. If you are a system programmer with authority code P, you can test nonprivileged system programs. These programs can be dynamically loaded from any one of the three major libraries. If the program is loaded from either JOBLIB or USERLIB, it is assigned to private, read/write storage. The attributes of public, read-only, system, and privileged are overridden. If it is loaded from SYSLIB, only the public and read-only attributes are overridden. As a result, you get a private copy of any module dynamically loaded from SYSLIB. Privileged modules so loaded remain privileged and write/fetch protected. This provides continued protection for the privileged routine.

Because of your authority code D:

- You may use all PCS commands in testing your nonprivileged programs.
- You may use symbolic addressing to display or dump any privileged CSECTs which have been dynamically loaded.
- You may display or dump the contents of your task's virtual storage.

If you are a privileged system programmer (authority code O), any module you dynamically load is assigned to private read/write storage. Only the attributes of public and read-only are overridden by the dynamic loader.

Your PCS capabilities with respect to privileged programs are the same as they are for a nonprivileged system program (except that the AT statement cannot be used in a privileged program). In addition, you can display, dump, or set IVM. You must exercise extreme caution in setting IVM, particularly in a multiprocessing environment, since other CPUs may be using the coding you are setting.

You cannot use other shared coding in the system for PCS testing, since it is not part of your virtual storage.

The PCS commands and their functions are discussed in detail in the Command System User's Guide.

TIME SHARING SUPPORT SYSTEM (TSSS)

For a complete definition of TSSS, its command language, and its modes of operation, see Time Sharing Support System, GC28-2006. The following discussion is intended only to introduce you to its facilities and use.

TSSS has two parts, a Resident Support System and a Virtual Support System. The Resident Support System (RSS) is very nearly independent of TSS; when it is invoked, TSS activity is temporarily suspended and RSS has access to all real storage and to the virtual storage of all current tasks. The Virtual Support System (VSS) performs the same basic functions as does RSS; however, it is invoked within an active task, relies on the TSS resident supervisor, and is time-sliced.

TSSS is a maintenance tool which has been developed for the system programmer (authority code 0 or P). It is independent of machine configuration, and, depending on mode of operation, it may be activated by pressing the CPU external interruption key, either from a remote terminal location or from predetermined points within a task during TSS execution.

When you are using RSS you are referred to as a Master System Programmer (MSP); when you are using VSS, you are referred to as a Task System Programmer (TSP). There can be only one MSP at any given time, whereas there can be more than one TSP, but only one per task.

TSSS has its own command language; it is constructed from the following elements:

Commands
Symbols
Literals
Operators

Commands

The following is a list of the commands that you can use with TSSS:

- AT - Places a linkage to TSSS at a specific point during TSS execution. The AT command must be followed by at least one other TSSS command, the one that will be executed when the specified location is reached.
- CALL - Provides access to command statements that are on tape or in the card reader.
- COLLECT - Accumulates data into a selected data field.
- CONNECT - Joins a TSP to a conversational task.
- DEFINE - Create temporary SP symbols.
- DISCONNECT - Removes TSSS capabilities from a terminal that is dedicated to an MSP or TSP.
- DISPLAY - Displays data at your terminal.
- DUMP - Displays data on an output device. (You must establish the output device prior to issuing a DUMP command by placing its address in the $DOUT data field.)
- END - Terminates the reading of command statements from a device and returns control to your terminal.

- IF - Designates that the execution of subsequent commands is conditional.
- PATCH - Temporarily overlays the contents of a data field. Its previous contents are saved.
- QUALIFY - Defines or changes the implicit meaning of subsequent operands.
- REMOVE - Deletes patches, and implanted ATs and their associated statements.
- RUN - Returns control to TSS without disconnecting your terminal.
- SET - Inserts data into a specified data field.
- STOP - Returns control to your terminal after statement execution.

Symbols

You may use the following types of symbols with TSSS:

- External Symbols - Refers to the actual (real or virtual) storage address of a data element.
- System Symbols - Refers to and qualifies storage areas. The first character of each system symbol is a $.
- SP Symbols - Defines and assigns a symbolic name to a data field. The data field defined by an SP symbol may exist in the system programmer's working storage or it may be a TSS data field.

Literals

You may use the following types of literals in the TSSS command language:

Decimal Integer
Hexadecimal
Character

Operators

You may use the following types of operators in TSSS command statement operands:

Arithmetic
Relational
Boolean

TERMINAL ACCESS METHOD (TAMII)

The following is a short overview of what TAMII is and how it is organized.

What is TAMII?

TAMII is a package of modularly designed programs for providing a user-directed, device-independent, terminal-computer interface. It provides both device dependent and independent functions for system and application programs.

TAMII provides a system programmer with a concise and well-defined interface for adding both new devices and/or new device function.

TAMII was designed and written to do the following for TSS:

(1) Improve human factors by

- providing a user-terminal interface under almost direct control of the user, through user commands and defaults
- use of buffering on input and output
- better user oriented error recovery; error recovery interacts with the terminal user to correctly handle recoverable errors.
- increasing communications line reliability due to better error recovery.

(2) Increase system performance and throughput (via buffering on input and output)

(3) Increase maintainability, reliability, and extendability by separation of function, modularity of code, and table driven device support.

TAMII provides the following basic services for the system and application programs, and users:

Establish, terminate, and control access between tasks/application programs and communication lines

Move data between application programs and communication lines

Maintain a device and data independent interface between task/application programs and communication lines

Establish and maintain a well-defined interface between device dependent modules and common system provided routines

Permit tasks, application programs and users to share communication lines, controllers, and terminals

Permit monitoring and altering of the telecommunications network

Handle device dependent and independent requests interchangeably

Complete input and output buffering transparent to the application program but under direct control of the user

Reliability, availability, and serviceability aids to assist in maintenance and extending device-function support

Place the user's communication environment under the user's direct control

Allow a priority sequence of interrupt processing, by an application, for operations to and from a terminal. TAMII supports the following types of interrupt processing:

Device 'EXITLIST'	ASYNC
Application program general 'EXITLIST'	ASYNC
SIR & DIR interrupt queuing	ASYNC
FINDQ work polling capability	SYNC
'CHECK' capability	SYNC

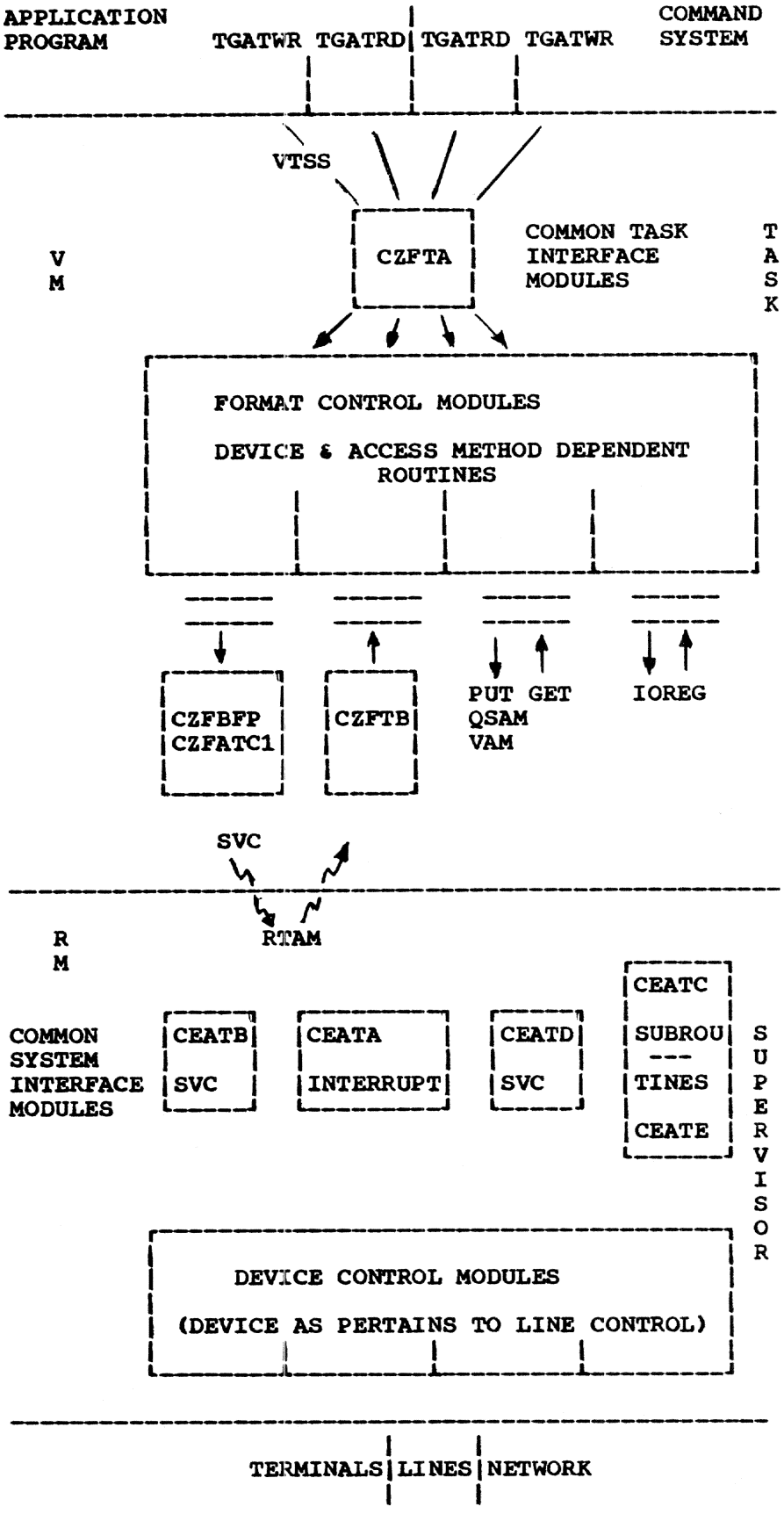


Figure 13. Overview of TAMII Organization

COMPOSITION OF TAMII

TAMII consists of four main components: two residing in the resident supervisor and two residing in the tasks initial virtual storage. The resident supervisor components are RTAM (Real Terminal Access Method) and a set of DCMs (Device Control Modules). The virtual storage components are VTSS (Virtual Terminal Support System) and a set of FCMs (Format Control Modules). See Figure 13.

RTAM -- Real Terminal Access Method

RTAM contains six common system control modules. RTAM controls all interaction between the system and the DCMs to provide one common interface between RTAM and the system, and between RTAM and the DCMs.

The following are the RTAM modules and their associated functions:

1. CEATA - interrupt handler, I/O completion and I/O request queue initiator for non-shared lines

2. CEATB - ATCS SVC handler; handles all task I/O requests, and supervisor I/O requests. CEATB validity checks the request, generates a request element (a buffer) for the request, queues the request on the pending request queue in its correct priority position, and calls the DCM to process the request, if required

3. CEATC - subroutine pool one; contains common subroutines used by both RTAM and the DCMs

4. CEATD - environment SVC handler; handles the SAVBFP, RSTBFP, and SETTCT requests:

   SAVBFP - saves all pending and not active I/O requests

   RSTBFP - restores to the pending queue previously saved buffers

   SETTCT - sets, resets and interrogates flags in the TCT for the task

5. CEATE - subroutine pool two; extension of CEATC

6. CEAR4 - MTT related SVC handlers:

   CONN - connect an MTT task to the system and make known a 'BEGIN' application name to the system

   DCON - disconnect an MTT task from the system and delete the application name from the 'BEGIN' table; also used to inhibit (ILOGON) and permit (PLOGON) BEGIN requests for an application task

   CKALOC - performs two functions:

      Mark a terminal under task control for IOREQ use (TAMII gives up control of the terminal), and release a terminal from task control and return control to TAMII

      Force communication line initialization for network control

   ATTACH - used to logically attach a separate function to a currently active terminal

LCONN - connect requested terminal to task under TAMII control

UFLOW - svc is used to interrogate and set the allowed count of users for the system and MTT application programs.

DCM - Device Control Modules

DCMs are supposed to be line controllers. It is the DCM's responsibility to get the data to and from the terminal. Ideally, the DCM has no knowledge of the actual terminal at the end of the communications link. DCMs are normally table driven from a Device Control Library Entry called a DCL.

It is the DCM's responsibility to handle the following functions when required:

Does final validation of all I/O requests

Builds the required channel programs to perform the I/O requests and initiate the I/O

Maintains line control during non-activity between terminal and task

Handles the initialization required for connecting a terminal to a task whether initiated by the user or the task

Sets up device dependent information in the required system control blocks

Handles all device dependent interrupt status other than channel end/device end and system PCI chaining requests

Provides error recovery for all abnormal endings

Handles device dependent timer routines

Provides simple output edit capability for supervisor messages to the terminal user

Determines length and type of input and checks user's input for user and hardware function requests (cancel, attention, etc.)

The following DCMs have been implemented:

CEDM01 - contains two functions which are used by CEDM02, CEDM03 and CEDM04.

CED1R - an input data interrogator. CED1R handles input length determination, special control character function handling (e.g., ATTENtions, line cancel, device control command determination and input buffering control function), and SOLICIT macro support.

CED1E - handles all error recovery and other unexpected or abnormal status completions for I/O requests.

CEDM02 - handles 2741 support.

CEDM03 - handles ASCII terminal support, specifically TTY33 and TTY35.

CEDM04 - handles 3215 operator console support.

CEDM07 - handles 3270 support. CEDM07 does not use CEDM01, but does its own input determination and uses CEDM09 for error recovery .

CEDM08 - handles the 3066 console support for the M168 operator console CEDM08 uses CEDM09 for error recovery.

CEDM09 - error recovery for CEDM07 and CEDM08. CEDM09's error recovery is much simpler than the error recovery used for CEDM02 and CEDM03.

RTAM Control Blocks

CHAATCS - virtual memory parameter list for requesting RTAM to perform some I/O request. The parameter list is pointed to by the executed ATCS SVC (SVC X'DB').

CHABFP - I/O request control area and data area. Contains information in the header area describing the requested operation. The data area contains the user's data for output operations.
CHABFPs are chained forward and backward, in a circular chain. The first entry in the chain is pointed to by the TCT (TCTBUF).

CHADCLE - device control library entry. The CHADCLE is a read-only control block used to drive the Device Control Modules and to set up the initial terminal environment at initial connection time. The DCLE's field definitions are dependent only on those DCMs that are to reference it.

CHAERR - a dsect covering the error recovery error report built to record permanent and intermittant errors for VMEREP handling.

CHAMTS - system control block used for controlling user access to the system or to special application tasks. The MTS contains a pointer to the segment table which contains the shared translate tables.

CHARMSG - the RMSG is RTAM's message file. The RMSG contains all of the messages used by RTAM for communicating with a user.

CHASCN - the SCN is a SYSGEN built table describing all I/O devices on the system. The SCN is used by RTAM as a base pointer for RTAM control block chains.

CHATCT - terminal control table entry. The TCT is RTAM's main control block. Every other chain or control block used by RTAM is headed in the TCT. The TCT is also used to maintain line status information for connected lines.
The TCT is pointed to by CHASCN (SCNTCT) and by the owner task's TSI if there is one (TSITCT).

CHATII - task interrupt information block. The CHATII is a dsect covering the CHABFP after the BFP has been reformatted for task I/O completion posting.

CHATIO - terminal I/O control block. The CHATIO describes the I/O currently in progress on the line. The TIO dsect covers two separate control blocks.

(a) The main TIO block contains the channel program needed to perform a requested operation. The TIO is pointed to by the BFP which describes the I/O request being performed. For certain line control operations, there may not be a BFP. The currently active TIOCB is pointed to by the TCT (TCTTIO) for the line.

(b) The second control block is an error recovery status save area. It is pointed to by the TIO which contains the error interrupt.

CHATRAN - shared translate tables. The CHATRAN dsect covers the translate tables which are shared between virtual memory and RTAM. They contain tables for performing line code to EBCDIC and back translation besides tables for folding and reverse folding EBCDIC codes.

CHATSI - task status index. The TSI is the main resident task control block. The TSI contains a pointer (TSITCT) to the TCT for the task's SYSIN/SYSOUT. The TSI also contains a pointer to a list of connected terminals for MTT applications.

VTSS - Virtual Terminal Support System

VTSS provides the common virtual storage interface between TAMII and the application program and/or task. VTSS handles all the user's requests for connecting/sending/receiving data, and disconnecting a logical terminal. VTSS attempts to provide complete logical device support for the application programmer, and consists of eight modules:

CZFTA - I/O request macro handler. Using the user's environment, the macro request code, and information about the FCM from the FCL, it determines what sequence of requests must be done to fulfill the request.

CZFTB - I/O completion interrupt handles; handles synchronous and asynchronous interrupts from RTAM for the task. Also does all processing needed for the 'EXITLIST' support and initiates the virtual storage connection process when a user connects to a task or an MTT application program residing in the task.

CZFTC - default extractor; gets defaults and sets the appropriate flags and fields for both TAMII and the user's environment. For O and P authority users, also handles the VSS device support default at logon time.

CZFTD - connect and disconnect; called to connect and disconnect a user and/or the user's SYSIN/SYSOUT components for TAMII by building the required control blocks.

CZFTE - terminal profile handler; handles the merging of the user's requested terminal session environment with the terminal's format requirements. Also handles the saving of the user's environment for profile processing.

CZFTG - device control command processor; handles all processing for the user's entered device control commands (screen commands).

CZFTP - common VTSS psect.

CZFBFP - subroutine pool; contains buffer allocation routines, ATCS parameter list build routine, and other common routines.

FCM - Format Control Modules

Format Control Modules are responsible for translating a user's output data stream into a form and sequence which the end device will accept and act upon in the way the user expects. On input, the Format Control Module is responsible for removing all device control information and setting up the data into an EBCDIC stream for the using program. The FCM is set up to handle a class of devices or access methods; e.g., CZFM00 handles all interactions with datasets.

It is the FCM's responsibility to handle the following functions when required:

1. For output --
   - edits output data against user function table
   - does any block or record formatting required
   - handles any physical line length limits and any required control character sequences
   - translates output data to line code
   - invokes correct routine to do I/O
   - checks return codes and sets up correct return code for the calling module
2. For input --
   - translates input data
   - removes any block and-or record format headers, etc.
   - deletes any device control characters
   - edits input data against user function table
   - moves input data to correct input area
   - checks return codes and sets up correct return codes for the calling module
3. Control requests; performs control function by either continuing the calling sequence or exercising its own code.
4. Maintains correct sequence and buffer links for buffered requests in virtual storage.
5. Handles any associated functions required for support of owned devices; e.g., conversational buffer.
6. Performs any special initialization which may be required for connecting a device.
7. Performs any special processing which may be required for disconnecting a device.

The Format Control Modules currently supported are:

CZFM00 - handles all operations with datasets. CZFM00 supports the following dataset organizations: QSAM, VSAM and VISAM (region and non-region).

CZFM01 - handles all supported hard copy terminals.

CZFM05 - handles the conversational buffer used for support of display terminals. CZFM05 is device independent. For actual device dependent activities CZFM05 calls a second level of FCMs to provide the actual device dependent support:

CZF3270 - local 3270 support
CZF3066 - 3066 Console support

VTSS Control Block Definition and Setup

CHAMTT - Virtual Memory Terminal header control block. CHAMTT is contained in CZFTP and is the head for the task's terminal and user tables. CHAMTT contains the following table headers:

(1) List of user's tables - if MTT/MUT type of task

(2) List of RTAM-owned terminals connected to the task.

(3) List of non-RTAM-owned terminals and pseudo terminals connected to the task.

(4) MTT work area pointers and saved work area pointers for exit from the MTT state.

List (1) above is organized by user number. The USN parameter on all the TAMII macros is used as an index into this list to get to the User's Terminal Control Block (CHAVTCB). See Figure 13.

Lists (2) and (3) above are essentially the same. They are organized by Relative Line Number (RLN). The RLN is assigned at connect time by RTAM for RTAM-owned terminals or by the Connect Module (CZFTD) if the terminal is not supported by RTAM. The RLN is a halfword number with zero being valid and a X'8000' denoting an unassigned RLN entry. Bit 1 (X'4000') determines which list the RLN pertains to. RTAM list -- list (2) -- bit 1 is a 0; non RTAM, bit 1 is a 1. For further information, see the CHATERM description.

The MTTBFP pointer determines which work area is in use. For normal TSS there is only one work area, for MTT there is an expanded active work area assigned and the original TSS work area is pushed down by being saved in MTT... and MTTBFP being changed to point to the new area.

CHAVTCB - Virtual User Control Block. One CHAVTCB is assigned for each connected user of a task. The VTCB contains pointers to the FCLs (see CHAFCL description below and figure 14) for the user's SYSINs and SYSOUTs. The corresponding default SYSIN and SYSOUT FCL pointers are maintained in the CHAVTCB.

CHAFCL - Format Control Library entry. The FCL contains the information needed by the Format Control Module for handling the user's SYSIN/SYSOUTs. This information pertains both to the user's environment profile and to the actual hardware characteristics of the SYSIN/SYSOUT mode (terminal or dataset). The system contains a set of FCLs for all supported devices and access methods. The system FCLs also contain a set of system defaults for the user's terminal environment which

are used until the user's profile can be accessed. At LOGON time the user is assigned a copy of the system's FCL. After LOGON has completed, the user's own environment profile is merged with the device requirements and limitations and is contained in the FCL.

CHATERM - terminal RLN to FCL translation entry. There is one CHATERM entry for every assigned RLN whether its for an RTAM or non-RTAM terminal. The CHATERM entry contains pointers to the FCL(s) used to define the SYSIN/SYSOUT connection between the RLN and the user. Also maintained in the TERM entry are the completed work flags for use by FINDQ when running an MTT application program.

CHAGMA - GATE macro parameter list. All of the TAMII macros build a fixed length format parameter list. This macro parameter list is described by the dsect CHAGMA.

TAMII CONTROL BLOCK ORGANIZATION

VTSS - User Macro to SYSIN/SYSOUT Translation

Each user (see Figure 14(A)) connected to a task is allowed a maximum of three SYSOUTs and three SYSINs active at any one time. One SYSOUT and one SYSIN are considered the default component. Each of these SYSOUT/SYSINs are described by a FCL entry. There is only one FCL for each device connected to the task. An FCL entry may describe only one SYSOUT or one SYSIN or it may describe both a SYSOUT and a SYSIN for the same device .

When the application program or the TSS Command System issues a TAMII macro, the user and the component for which the request is directed is specified using the USN, CPO and/or CPI parameters. If the USN and CPO and/or CPI are not used, a USN of 0 (the task owner) and the default components for that USN are used. Using the USN number, CZFTA computes a pointer to the user entry in CHBMTT. From this entry CZFTA picks up the pointer to the user's VTCB which contains the pointer to the FCLs for the user. If a CPO or CPI had been given, CZFTA uses it as an index into the list of SYSIN/SYSOUTs owned by the user. If not given, CZFTA uses the default pointer from the VTCB to get the correct FCL.

VTSS - RTAM Interrupt to FCL and VTCB Translation

Each interrupt from RTAM (see Figure 14(B)) which is received by the task's RTAM Interrupt Processor (CZFTB) contains a halfword number called a RLN - Relative Line Number. The RLN is used by CZFTB to index into a table of 'term' entries pointed to by CHBMTT. The TERM (CHATERM) contains the user number (USN) of the user owning the terminal and pointers to the FCLs for input and output. If needed, the VTCB address is loaded from the associated FCL.

RTAM - Virtual Memory Request to TCT Translation

When VTSS sends an I/O request to RTAM (see Figure 15(A)) by executing the ATCS SVC, VTSS fills in the field ATCSRLN from FCLRLN. The RLN is used by RTAM to determine the device to which the request is directed.

Upon receipt of the request, RTAM picks up the RLN from the parameter list. If the RLN is zero, RTAM assumes the request is for the task

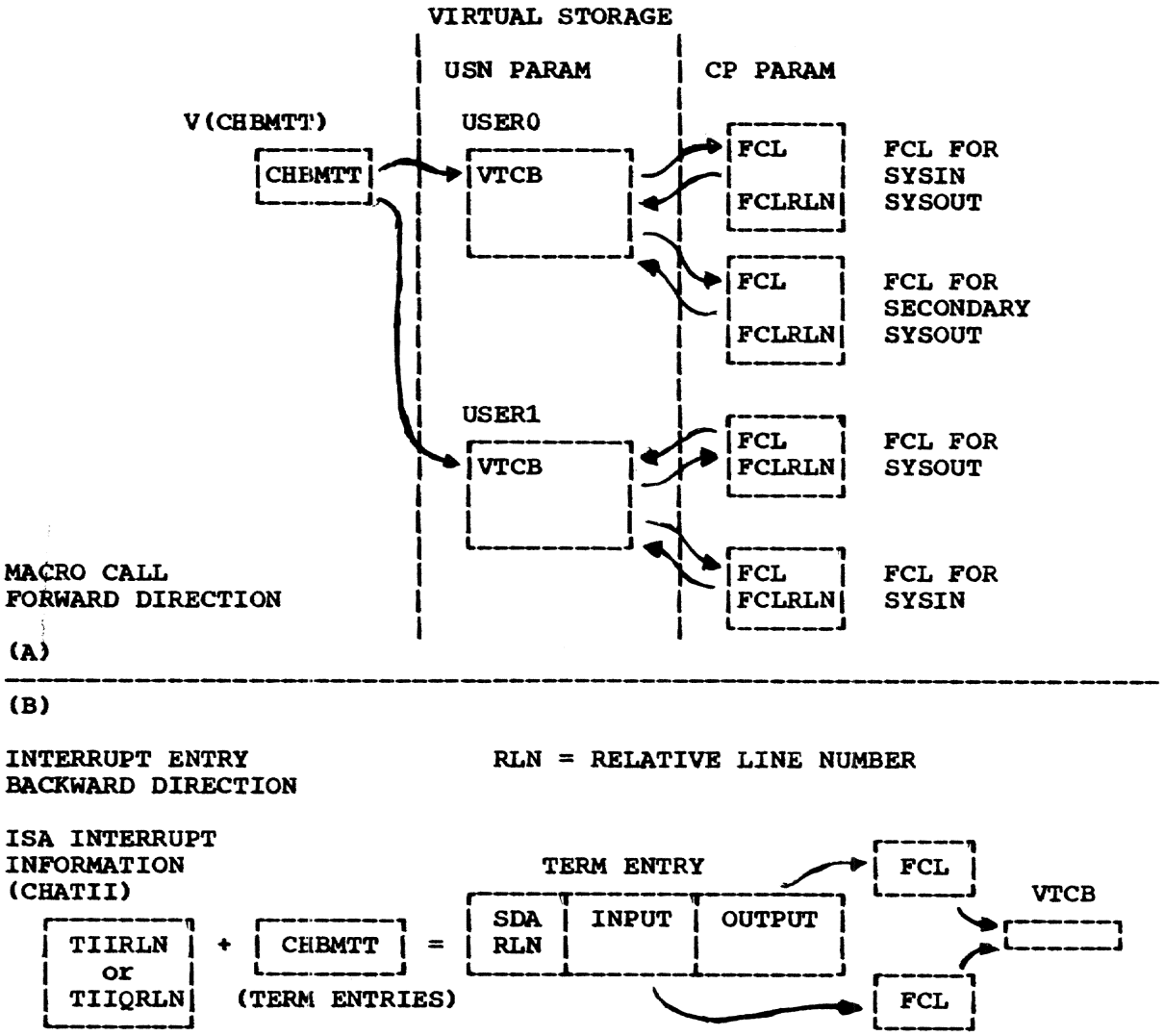


Figure 14. TAMII Control Blocks - VTSS

owner's terminal and loads the TCT pointer from the TSI. If the RLN is non-zero, RTAM uses the RLN as an index into a list of fullwords pointed to by the task's TSI. The first halfword is a set of flags and the second halfword is the SDA of the device. The SDA is used to compute the address of the scan table entry, from which the TCT address is loaded.

RTAM - I/O Interrupt to Owner's Task TSI Translation

Whenever RTAM receives an I/O interrupt (see Figure 15(B)), the interrupt GQE contains the SDA of the line. RTAM uses the SDA of this interrupt, to compute the scan table entry. In the scan table entry is a pointer to the TCT for the line. If there is not a TCT pointer, RTAM assumes that the interrupt is an initial interrupt from a user who wants to connect to TSS. In this case, RTAM allocates a TCT and places its address in the scan entry. If there is a TCT pointer, RTAM tests the TCTLOG flag. If the flag is on, the terminal is in the logon process so there is no task and therefore no TSI pointer. If the flag is off, RTAM loads the TSI pointer from TCTTSI .

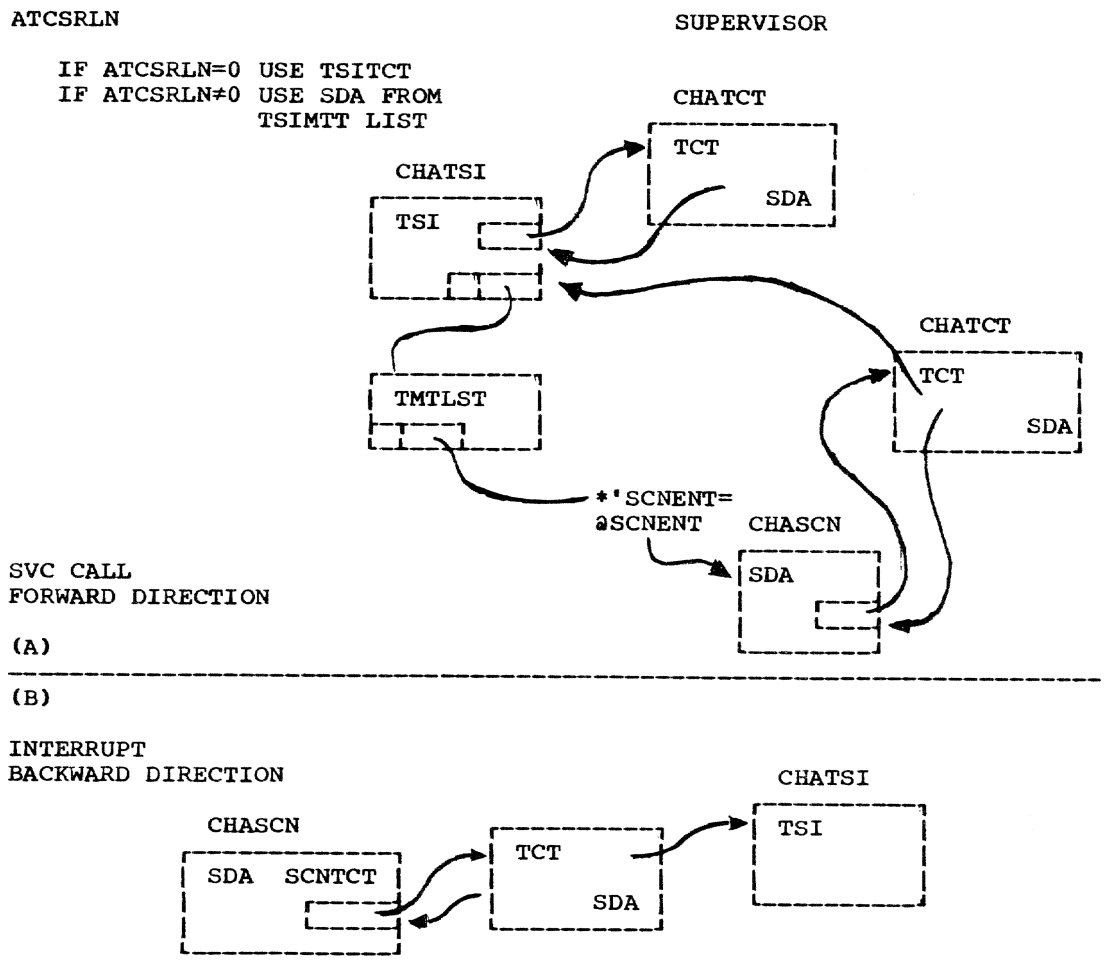


Figure 15. TAMII Control Blocks - RTAM

RTAM I/O Queue Organization

When VTSS issues an I/O request (see Figure 16), RTAM builds a request block called CHABFP (BFP for short) and moves all the pertinent information and data out of virtual memory and into the BFP. The BFP is chained in a pending request chain of other BFP blocks. This chain is in a priority order. Any request which has the break flag (GMABRK) set is assumed to be a top priority request and is chained at the top of the pending queue. Following these requests are the normal I/O regeusts issued by virtual memory. At the end of the request queue, are the Solicit and Buffer Command Read operations.

RTAM builds a channel program for a BFP in a control block called a Terminal I/O Control Block or TIOCB (CHATIO). The TIOCB is not constructed for a BFP until it is time to do the actual I/O. When the TIOCB is built the BFP is pointed to its associated TIOCB and the TIOCB is back pointed to the BFP.

When the TIOCB is activated by initiating the I/O, the TCT for the device is updated to point to the TIOCB. The pointer TCTTIO should always point to the current active TIOCB if there is one.

RTAM I/O Chaining

RTAM supports chaining (see Figure 16) of active I/O channel programs through a TIC and PCI interrupt operation. Any channel program which can be chained from is ended in two NO-OP CCWs and a pointer is set in the TIOCB to point to the second NO-OP which will become a TIC CCW.

When a TIOCB is added to the active I/O chain and is to be chained to, the second NO-OP of the previous TIOCB is changed to a TIC CCW. The TIC-to address is the address of the start of the new channel program and the second fullword of the TIC CCW is the address of the TIOCB being TICed to. When the set up of the TIC CCW is complete, the command chaining flag is set in the first NO-OP to activate the TIC CCW.

The reason for two NO-OPs is to allow chaining as late in the active channel program as possible. If the ending I/O transfer CCW were used, it would lengthen the 'miss window' by the amount of time it takes to transfer the data. In this way the window for missing is only the amount of time it takes the channel to process the NO-OP CCW.

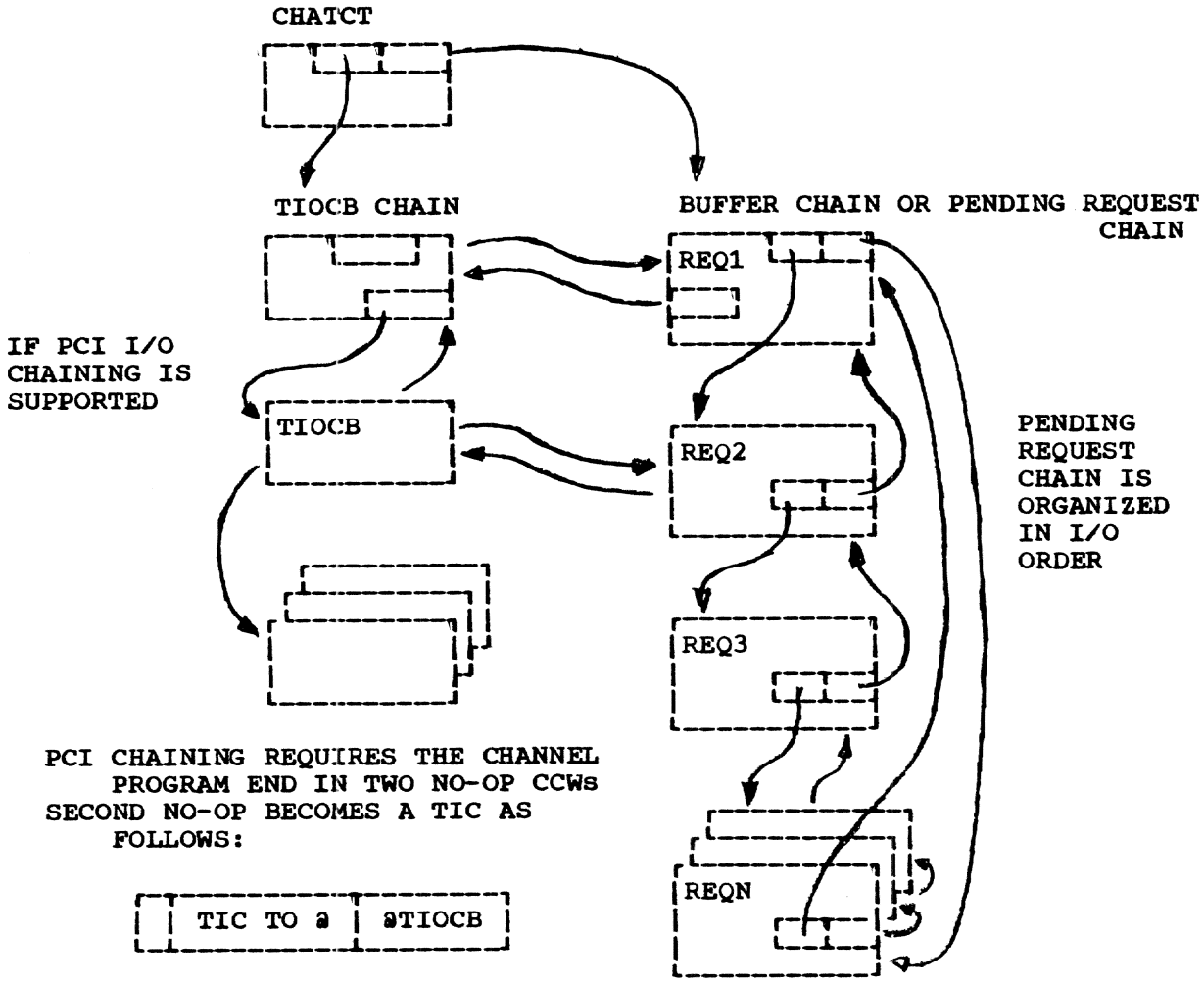


Figure 16. TAMII Supervisor I/O Queues.

NCP SUPPORT FOR TSS

The NCP facility of TSS provides support for a channel attached 3704 or 3705 Communications Controller, executing either the Emulation Control Program Level 3, or the Network Control Program/VS Level 5.

The following SNA devices are supported as TSS system SYSIN/SYSOUT components:

3767 Display Unit models 1, 2, and 3

3277 Display Unit model 2 using a 3271 Control Unit model 12 or 3274 Control Unit model 1C using SDLC

3278 Display Unit models 1, 2, 3 and 4 attached to a 3274 Control Unit model 1C or a 3276 Control Unit model 11, 12, 13, or 14 using SDLC

The following 3270 class printers using either 3270 data stream or SNA character string (SCS) data streams are supported:

3284 models 1 and 2

3286 models 1 and 2

3287 models 1 and 2

3288 model 2

3289 models 1 and 2

TSS NCP support allows the operator to activate (load and execute a 'bring up' sequence), deactivate, and dump a 3704 or 3705. The operator or an authorized user can run a line trace against an active SDLC line or an active 370X Communications Controller running NCP/VS Level 5.

ASSIGNING NCP CAPABILITY TO TSS

To support the NCP on TSS there are two new parameters added to the CLOP macro instruction -- SUBAREA and MAXSUBA. If they are not specified, TAMII's NCP function is deactivated and an error message will be given whenever an NCP activation is attempted.

MAXSUBA is the maximum number of subarea numbers to be assinged in the network. It is specified as a decimal number which is a power of two, minus 1 (e.g., 3, 7, 15, etc.) but not 0 or 1; subarea values of 0 and 1 are reserved for TAMII.

SUBAREA is the number to be assigned to RTAM as its subarea address value. It must be greater than zero and less than MAXSUBA. A value of 1 is recommended for use with NCP Level 5.

TSS Restrictions for MAXSUBA and SUBAREA

TAMII considers the SDA to be a network address belonging to subarea 0. Therefore, the number of bits used for a subarea number must leave enough bits to accommodate the installation's largest SDA.

TAMII restricts the number of minor nodes belonging to a subarea to 512 or less. This is due to the size of the tables moved to the supervisor and the nechanism used to move the tables. The recommended value for MAXSUBA is 127 which allows the full 512 elements per subarea and 128 subareas. It also allows SDA values to X'1FF'.

DEFINING AN NCP FOR TAMII

Not all of the information coded in the NCP macro instructions is used by both TAMII and the NCP generation routines. However, all of the macro instructions should be coded with the possible needs of both TAMII and NCP generation in mind.

The following is a description of the macro instructions and their parameters; some are NCP and TAMII parameters, and some are for TAMII only. (Refer to the SYSGEN for NCP manual, IBM Order Number GC30-3008-5.)

> Note: there must be no discrepancy between the source used for the NCP generation and the source used for the TSS NCP table generation; the source can be used for both generations without modification.

The PCCU macro

AUTODMP={YES|NO}
: specifies whether, after an unrecoverable failure of the communications controller or the NCP, a dump of the communications controller storage is to be taken prior to an automatic reIPL (i.e., AUTOIPL=YES). If AUTODMP=NO and AUTOIPL=YES the NCP is reIPLed; if AUTOIPL=NO also, the NCP is deactivated and removed from the system.

AUTOIPL={YES|NO}
: specifies whether after an unrecoverable failure of the NCP or communications controller and after the dump (if one is taken), a fresh copy of the NCP is to be automatically loaded into the communications controller and restarted. If AUTOIPL=NO, the communications controller is deactivated and removed from the network. If the communications controller is successfully reloaded, configuration restart attempts to return all resources to the state prevailing at the time of the failure. However, all tasks will have been disconnected and abended. All dial communications will have to be redialed.

AUTOSYN=NO
: TAMII does not support auto-synchronizing.

INITEST=NO
: TAMII does not support the initial test routine load.

The rest of the PCCU parameters are ignored by TAMII.

The BUILD macro:

OLT=NO
: TAMII does not support TOLEP.

MAXSUBA=n
: discussed previously. Code MAXSUBA=3 or greater. The NCP default value of zero must not be used. Also, MAXSUBA value should be the same for all NCPs and the TSS SYSGEN.

The HOST macro:

When coding the following parameters, note the restrictions placed on their use by TAMII.

BFRPAD=0
TAMII requires zero and does not support leading pad characters.

MAXBFRU=count
specifies the number of buffers that the host processor allocates for each data transfer (channel program) received from the local communications controller.

UNITSZ=length
specifies the length of a host processor's buffer. For TAMII, the value of MAXBFRU multiplied by the value of UNITSZ must be 4000 or less. Recommendation: MAXBFRU=8 and UNITSZ=256.

STATMOD={YES|NO}
TAMII supports either value. Recommendation: STATMOD=YES.

The GROUP, LINE, PU and LU macros:

BUFLIM=n
the number of 64 byte blocks RTAM will allocate to hold the transmissions to be sent to a specific LU; 'n' is a decimal number from 0 to 256.

ISTATUS={ACTIVE|INACTIVE}
specifies whether a node is to be ACTIVE or INACTIVE when the 'owning' node is made ACTIVE.

Example: for an NCP gen such as the one shown immediately below, the ISTATUS shown in parenthesis is the assumed value; whenever ISTATUS is not specified for a node, that node assumes the ISTATUS value of the 'owning' node:

```
          GROUP  ISTATUS=ACTIVE
L1        LINE                          (ISTATUS=ACTIVE)
P1           PU  ISTATUS=INACTIVE
U11            LU  ISTATUS=ACTIVE
U12            LU                       (ISTATUS=INACTIVE)
P2           PU                         (ISTATUS=ACTIVE)
U21            LU                       (ISTATUS=ACTIVE)
U22            LU                       (ISTATUS=ACTIVE)
```

In the example above, when the NCP is activated, TAMII will also activate the following resources: LINE-L1, PU-P2,LU-U21 and LU-U22, because each higher level 'owning' resource has also been activated. PU-P1, because its ISTATUS=INACTIVE, is not automatically activated and therefore its LUs are not activated even though LU-U11 has ISTATUS=ACTIVE. Now when the operator activates the PU-P1, then TAMII will also activate the LU-U11, but not LU-U12 because its ISTATUS is INACTIVE.

With TAMII, ISTATUS=INACTIVE means the resource must be activated by direct operator command.

MODETAB=device type code
MODETAB on an LU is used by TAMII to specify the type of device the LU is; only the following codes are accepted:

3278 3284 3287 3277 3767 3286 3289 3288

VPACING={(n[,m])|0}
defines the way TAMII and the NCP are to pace the flow of data between the host processor and the NCP for sessions with the associated logical units.

n -- specifies the number of messages that TAMII is to send to the NCP before waiting for a pacing response; 'n' is a decimal number between 1 and 255 and must be equal to or greater than the value of the 'n' used with the NCP's corresponding PACING paramter.

m -- specifies which of the 'n' (the parameter above) requests will be flagged to request a pacing response from the NCP. TAMII sends at most, n-m additional data requests if a pacing response is not received. If 'm' is not coded, 'n' is assumed. Specify 'm' as a decimal number between 1 and 'n', and must be equal to or greater than the value of 'm' used with the PACING parameter.

0 -- specifies that no pacing is to be performed for sessions with logical units associated with the macro instruction in which VPACING is coded.

Defining the NCP Network to TAMII

The telecommunications network is defined to TAMII by preparing and assembling definitions of the two types of major nodes and then filing the object modules in joblibs owned by the operator.

The two types of major nodes supported by TAMII are:

1. NCP major node
2. Switched SNA major node

An NCP major node consists of a 3704 or 3705 Communications Controller (locally attached), the network control program (NCP) being executed in that controller, and the physical configuration defined for that NCP during NCP generation.

Switched SNA major nodes consist of either or both of the following supported SNA terminals:

3767 Communications Terminal
3274 or 3276 Display Controllers and Displays

Switched SNA major nodes do not include the switched SDLC links to which the terminals are attached (these lines are part of the NCP major node). The minor nodes of a switched SNA major node are the SNA controllers and their associated logical units.

Any number of definitions can be prepared, assembled and filed, to represent combinations of major nodes that may be desired in an active network under different circumstances. Each major node definition contains statements defining all minor nodes encompassed by the major mode. Each major node must be assigned a unique name and must have an entry in the TSS*****.SYSRCS dataset (discussed in detail later) in order for TAMII to be able to use the major node.

Defining the NCP and Remote Terminals

One or more NCPs for each communications controller must be generated and stored in a joblib. The source deck used to create each NCP is then assembled using the TSS macro libraries. The object modules created by

assemblies are stored in either the same joblib or a different one from the NCP load modules. The information as to what is stored, and where it is stored, is conatined in the SYSRCS dataset in the region whose name is the same as the resource name. The name assigned to the region that contains these definitions is the name by which TAMII will recognize the NCP.

Defining Switched SNA Major Nodes

A switched SNA major node is defined by a single BUILD macro statement for the major node and separate PU and LU statements for each minor node. One BUILD macro with TYPGEN=SWNET must be included in each source dataset, placed before the first PU macro. The BUILD macro assigns a subarea value to the major node for TAMII's use in assigning addresses to the minor nodes.

The PU and LU macros define physical units and logical units attached by switched SDLC lines. The PU and LU macros are the same as those used to define a NCP major node.

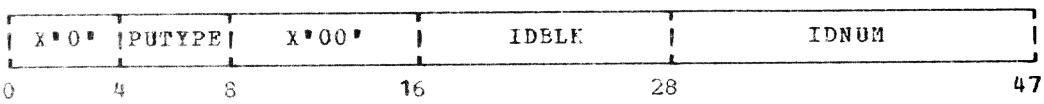
The installation may define multiple sets of switched SNA devices. This allows the network operator to selectively activate a subset of all the switched SNA devices using the ACTIVATE command. However, all major and minor node names known to TAMII at any one time must be unique.

If contact is to be established with a physical unit by means of a 'dial in' operation, the unit's station identification number must be placed in the SYSRCS region called BIRCNAME, followed by the physical unit's resource name in the format: '

'major node name.physical unit name'

When the physical unit dials in, TAMII searches the SYSRCS region BIRCNAME looking for the station id. If found, the physical unit's name is retrieved and if needed, the major node containing the definition is activated automatically. The information from the physical unit definition is then used to complete the connection process.

The unit's station identification nuber is a 48 bit number which is unique for each station within the network (not just within the major node). The station id is structured as follows:



PUTYPE - the physical unit type as follows:

3767 - 1

3274 - 2

3276 - 2

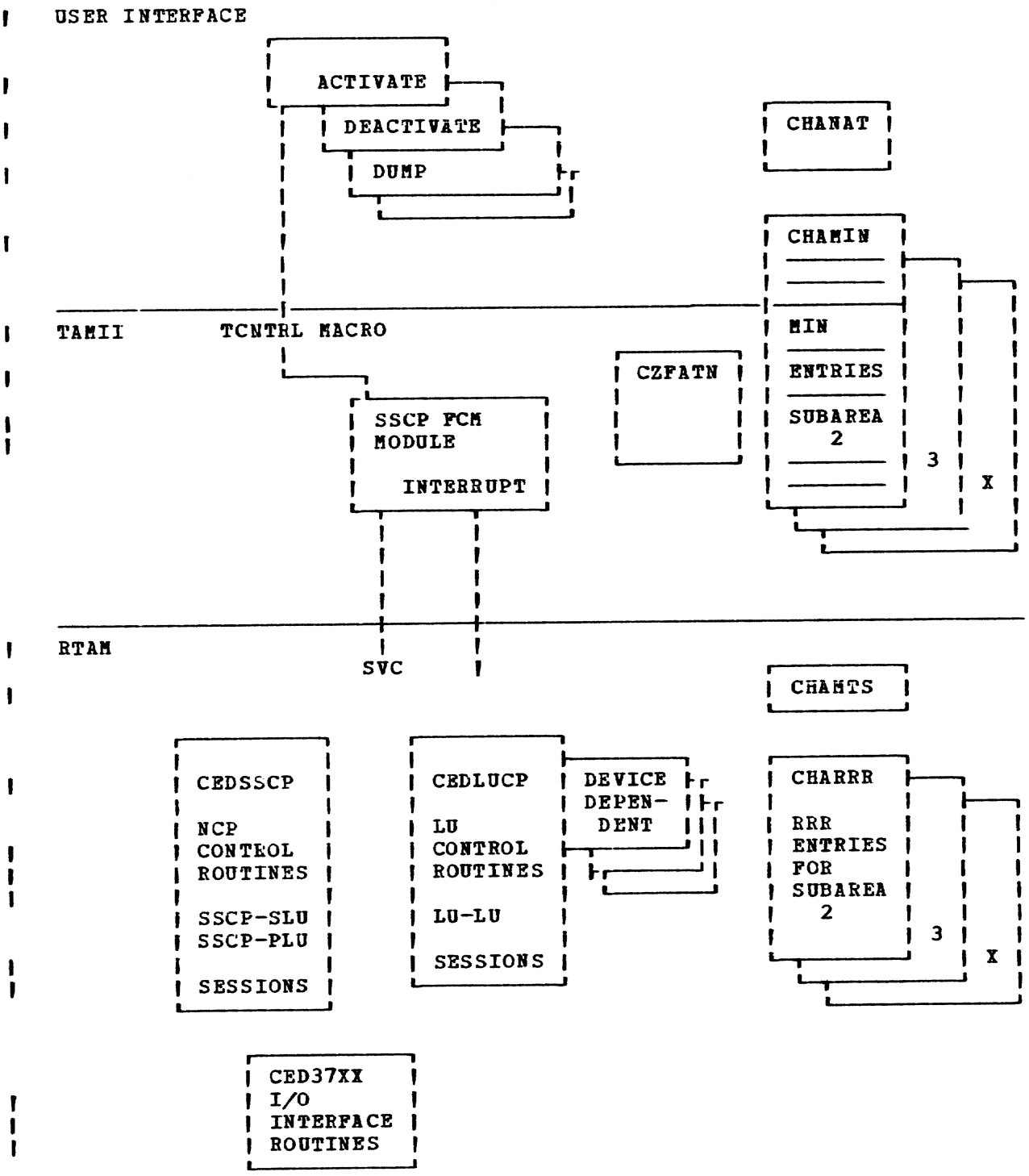
IDBLK - the 12 bit binary block number assigned by IBM to the specific device

IDNUM - the 20 bit binary identification number assigned to the station being defined.

ORGANIZATION OF TAMII NCP/SNA SUPPORT

The TAMII NCP/SNA support consists of three groups of modules. The first group consists of the user interfact routines. These routines convert the user's requests into control requests for the NCP. The second group reside in the virtual memory portion of TAMII and consist of two routines. One, CZAPTN, is the NCP network path translation routine. The other routine is the PCM used by the user control modules to communicate with the NCP control routines residing in RTAM and the NCP. The third group of modules reside in RTAM in the resident supervisor. These modules control all communications with the NCP and any supported SNA devices.

When a TSS NCP gen is performed, a set of tables is created which contain the definition and path information for the NCP. As part of the NCP activation process these tables are read into shared virtual memory and are used to fulfill the function of the SDAT (CHASDA) in virtual memory and the scan table (CHASCN) in real core. The virtual memory table (CHAMIN) is called the minor node table -- MIN. It resides in shared virtual memory and is connected and disconnected as needed. The real core table is called th Resource Resolution Table -- (CHARRR). It is read into virtual memory by the active process and moved to real core upon execution of the ADSBA -- Add Subarea SVC (discussed later in this document). The RRR is used by RTAM to control the activation, deactivation, and allocation of the NCP network resources.



As an NCP is activated, an entry is made into the Node Active Table -- NAT. The NAT is a dictionary maintained by the NCP user interface modules using the dictionary handler CZASD. CHBNAT is an IVM control block which contains the RSPI number and other information needed to connect and disconnect the NAT dictionary. When an NCP is activated a major node entry -- MJN (CHAMJN) is built in the NAT dictionary. The MIN and RRR are read into memory and a pointer, in RSPI format, to the MIN is placed in the MJN entry.

Using a TAMII OPNDST the NCP is connected to TAMII and the device is allocated to the task. With the completion of the OPNDST the system control blocks are setup to perform the activation function for the NCP.

The activation function consists of the loading of the 370X with the specified NCP load module and the activation of the NCP and its resources using the appropriate SNA commands.

TAMII Format Control Module Support for the NCP

CZFSSCP is the name of the Format Control Module used to support the NCP. It validates and handles the special TCNTRL type codes used to communicate between the virtual memory user control modules and the RTAM SSCP control module. CZFSSCP also contains special entry points for handling error records, line trace records, and dial in records returned by the NCP.

Normal read and write operations are not used or supported by CZFSSCP. Any attempt to use these requests results in a nonsupported (X'20') return code.

NCP PATHFINDING SUPPORT CONTROL BLOCKS AND HANDLER

For NCP Resource Resolution or as referred to in TAMII, NCP Pathfinding, there is an integrated set of control blocks. These control blocks all reside in shared virtual memory and except for the CHBNAT, header control blocks are disconnected when not in use. There are three control blocks used by the NCP modules in virtual memory. Except for CHBNAT, the control blocks are built by the NCP/TSS gen process and are loaded from datasets as part of the NCP activation process. The three control blocks are CHBNAT, MJN, and MIN.

CHBNAT is the anchor for the NCP control blocks. It is loaded as part of IVM and as such is addressable by all tasks in the system. CHBNAT contains system dependent NCP values such as the maximum subarea value, the subarea mask and the subarea shift value. CHBNAT also contains the anchor for the MJN dictionary. This anchor is in the format RSPI, RPN and page count, and is used whenever a task has to connect to the dictionary. Included in CHBNAT is the MJN lock word which controls access to the MJN dictionary.

MJN -- major node entry -- is a control block which describes the NCP control program to TAMII. It is built from the contents of the BUILD, HOST, and PCCU macros used in the gen of the NCP load module. MJN also contains information needed by the RESTART and DUMP commands. The MJN entry resides in a control block called the MJN dictionary. When the ACTIVATE command for the NCP is issued, a MJN entry is built and placed in the dictionary using the dictionary hanler module CZASD. Afterwards, the MJN entry can be retrieved by calling CZASD3 with the NCP name. The MJN for the NCP contains the address of the genned resources owned by this NCP -- the MIN table. This address is also in RSPI, RPN format. The MIN lock word is kept in the MJN; this lock controls access to the MIN.

The MIN -- minor node table -- is a gen created table of all resources owned by the subarea of the MJN. This table is basically the SDAT for the NCP or other subareas. There is one MIN entry in the table for each SNA resource defined in the NCP gen. A resource is the NCP control program (always resource 0), a communications line (LINE), a control unit on the line (PU macro) or a terminal or end user on the control unit (LU macro). Each resource has a unique name assigned at NCP gen time.

The path tables, MIN and MJN, are never connected to a task except under appropriate locks. The NCP Deact Command Processor assumes the tables are not in use if they can be write-locked and are therefore available to be FREEMAINed. The MJN and MIN locks are set up like data-

set RESTBL locks. The pathfinding module CZPATN uses CZCOH and CZCOI to lock and unlock the table locks.

RTAM/NCP SUPPORT

The Resident Terminal Access Method (RTAM) has a set of modules that provide the required NCP support. The modules were added in two areas: system support routines had CEATF (connect device) and CEATG (connect resource resolution tables) added; the device control modules had the LU-LU session control and SNA device support added. A description of these new modules follows.

System Support Routine Additions

CEATF (allocate device) is called by SVC from virtual memory to allocate a specific device to a task. Its parameter list is described by the DSECT CHALCN. The device address is passed to CEATF by an SDA or an RID. It locates the resource entry for the device and determines if the device is available. If the device is available, CEATF allocates and builds the necessary RTAM control blocks, sets up the required tables and calls the Device Control Module responsible for the device to complete the connect request. Upon return from the DCM, the allocation is complete and CEATF returns to the task with needed device information in registers 0 and 1. For device allocation failures, CEATF has a comprehensive set or return codes describing the reason for a failure.

CEATG (connect/disconnect resource resolution tables) is called by SVC from virtual memory to connect or disconnect the Resource Resolution Tables (CHARRR) which define the resources controlled by a major node in the SNA network. These tables are also used by RTAM for device allocation, control block anchors and status save areas. The RRR fulfills the same function for the major node and RTAM as the scan table (CHASCN) does for TSS and RTAM: the two are analogous. For a connect, virtual memory (during the activation process) issues the Add Subarea SVC (discussed later) with its associated parameter list. The parameter list contains the subarea address of the RRR which was previously loaded by the task and the count of entries in the RRR. CEATG validates the request and the subarea, allocates supervisor memory, and moves the RRR tables to the supervisor. For a disconnect, virtual memory issues the Add Subarea SVC with disconnect set, and the subarea address. CEATG checks the RRR before releasing to make sure it is currently not in use and if available, releases the supervisor copy.

Device Control Modules

CEDM0B fulfills two functions. First, it sets between the DCM's initiate request entry points and RTAM, to standardize the calling interface with the entry conditions expected by the other DCM entry points so that the RTAM work queue dispatcher is presented with a common interface across entry points.

The second function provided by CEDM0B is a primitive work queueing and dispatching mechanism for queueing work between different TCTs. This avoids the multi-cpu locking problems which would exist if one attempted to lock more than one TCT at a time.

CED0BQ is the enqueue entry point. A TCT and buffer address, along with a queue number, is passed to CED0BQ by the calling routine. The buffer must not be chained on any TCT at the time of the call. CED0BQ enqueues the passed buffer and TCT on the requested queue and returns. It is the dispatcher; it is called by CEATA, CEATB, and CEATD just be-

fore returning to their caller. It searches the queues for work following these simple rules:

The queues are processed in a FIFO order from queue 1 to queue 7.

CED0BD does not move from one queue to the next until all work on the queue has been processed.

If the TCT to be dispatched to is locked, CED0BD leaves the requests queued, and exits.

CEDM0B contains the queue headers, hard coded in the back of the module; there are seven queues:

Queue 1 is for error recovery; it has the highest priority.

Queue 2 is not in use.

Queue 3 is used for posting write completions. Queueing on the queue results in the buffer being sent to the write completion entry point of the DCM responsible for the TCT.

Queue 4 is used for posting read completions. Like queue 3 the responsible DCM's read completion entry point is called.

Queue 5 is used for requesting initiation of a function. Work for this queue is sent to the initiate request entry point of the DCM. This is the same entry point that processes SVC requests from a task.

Queue 6 is used for requesting I/O initiation. Requests queued here are sent to the DCM responsible for transmitting and controlling the I/O interface.

Queue 7 is the lost path and/or resource forced disconnect requests. These are processed last to ensure that all other requests destined for the lost resource have been removed from the queue. Any reuests on queue 7 are sent to the DCM's error recovery entry point.

CED37XX handles the I/O channel interface for all 370X type devices. It is set up to interface to the I/O side of RTAM and is called by CEATA on I/O interrupts from the 370X device. It is queued to by the NCP control modules CEDSSCP and CEDLUCP whenever they have output to be sent to the 370X control program.

CED37XX contains the routines used to load local 370X control programs. These routines handle the NCP or the EP program.

CEDSSCP is RTAM's system services control point module for controlling the TSS/SNA network. CEDSSCP handles all network control functions including the activating and deactivating of all network resources. It oversees all connections and disconnections of LUs with TSS and handles all network error recovery, recording and restarting, if required.

CEDLUCP is the RTAM LU-LU session control module; it handles all SNA protocol requirements needed for the support of the LU-LU session. It is device independent and relies on a sublevel of DCMs to handle the device dependent data manipulation requirements. CEDLUCP only supports the following session type: half duplex flip flop data flow within brackets.

CED327R is the device dependent DCM used by CEDLUCP to support remote 3270 devices; it handles all 3270 device functional requirements imposed by the SNA 3270. The corresponding DCL is CEDL70R.

ADSBA — Add/Delete Subarea SVC (SVC212)

The ADSBA SVC is used by the virtual memory NCP activate and deactivate routines to move the required subarea tables to the supervisor.

The ADSBA SVC parameter list is described by the DSECT CHAADSB.

The following codes are returned to the task in the task's general register 15 after execution of the SVC.

Code	Explanation
X'0'	successful add or delete
X'4'	subarea to be added already exists subarea to be deleted does not exist
X'8'	subarea to be deleted is still in use
X'C'	invalid parameter list; SVC not executed; SVC parameter list not on doubleword boundary, or VMA for parameter list does not exist
X'10'	subarea number is invalid; subarea number is 0, 1, or greater than the maximum subarea number allowed
X'14'	unable to allocate space to hold table
X'18'	the RRR header is invalid; the count of entries or the subarea number does not agree with the ADSBA value or the subarea mask value is not the same as the sysgened value

LCONN — Connect TAMII Terminal SVC Request (SVC205)

The LCONN SVC is issued by the TAMII virtual memory module to cause RTAM to allocate and build the necessary tables to connect a TAMII device to a task.

The LCONN SVC parameter list is described by the DSECT CHALCN. The SVC must be at the head of the parameter list and be executed. The parameter list must start on a fullword boundary.

The LCONN SVC processor returns the following information to virtual memory in registers 0 and 1:

Register 0 --
byte 0 -- zero
1 -- device type code from TCTDTY
2 -- zero
3 -- the Device Control Module index for the device

Register 1 -- the relative line number in bytes 2-3 by which RTAM knows the device

The LCONN SVC processor returns the following codes in the task's register 15:

Code	Explanation
X'0'	LCONN request was successful
X'4'	device is in use by the system
X'8'	device is in use by RTAM
X'C'	device is not an RTAM device
X'10'	resource entry for path is in use
X'14'	subarea has not been activated
X'18'	invalid SDA or RID given
X'1C'	RID from LCNRID is invalid
X'20'	no space available for TCT
X'24'	unable to allocate relative line number for device
X'28'	not used
X'2C'	invalid LCN parameter list

TSS*****.SYSRCS DATA SET

To activate any resource -- NCP, PEP, EP, etc. -- this dataset must exist.

Each region in this dataset will have the same name as the resource, and will contain information that pertains to the particular resource. Any resource information that is not defined in this SYSRCS dataset must be included as parameters in one or more of the NCP commands (discussed later in this section), with one exception; TYPE= must be defined in the SYSRCS dataset; it is not a parameter of any of the NCP commands.

Any parameter entered by the system programmer in any NCP command will override the same parameter if it is predefined in the SYSRCS dataset for this resource. A list of the parameters that may be predefined in the SYSRCS data set is as follows:

TYPE=
identifies the resource to TSS; specified as NCP, PEP, EP, or a 1-8 character name (the first character must be alphabetic) supplied by the installation.

DSNAME=
specifies the name of the dataset from which the resource load modules will be taken.

SUBTYPE=
identifies the type of hardware that will be used for this resource; i.e., 3704, 3705, etc.

MINNAME=
the name of the load module which contains the tables that will describe the resource to TSS. This parameter does not apply to an EP.

LOADNAM=
the name of the final load module that will run in the resource.

PHASE1=
the name of the initial load module that will 'bootstrap' in the second phase.

PHASE2=
the name of the second load module, called 'the second phase', which will read in the final load module.

ROUTING=
describes the different routes that can be used to get to the resource.

FCL=
the name of a TAMII control table that contains detailed information concerning the TAMII interface to this device. This table will contain the current status of the device.

RESTART=
tells TSS whether or not to perform an automatic restart if an error condition should occur.

AUTODMP=
tells TSS whether or not to perform an automatic dump if an error condition should occur.

DMPPH1=
the name of the initial load module that will be used to 'boots-trap' in the final dump load module.

DMPPH2=
the name of the final dump load module.

DUMPDS=
the name of the TSS dataset that will contain the dump output.

The SYSRCS dataset may be edited using the TSS Editor. The TSS datasets that must be available in order to activate an NCP are DSNAME= and TSS*****.SYSRCS.

ACTIVATION OF AN NCP/PEP OR EP

The activation of an NCP/PEP or EP entails the transfer of particular load modules from the TSS system to the NCP/PEP or EP, and the activation of those load modules in the resource. In order to perform this function certain types of information must be available to the TSS system. Each resource that is to be activated has some information that is unique. Activation of the resource is initiated by the ACTIVATE command.

DEACTIVATION OF AN NCP/PEP OR EP

The deactivation of an NCP/PEP or EP entails the severing of communications between TSS and the resource. In addition, all tables read in and generated during and after the activation of the resource are deleted from the system. The region in the TSS*****.SYSRCS dataset of the same name as the resource, will be scanned to determine the type of resource being deactivated as the deactivation processes differ.

DUMP OF AN NCP/PEP OR EP

The dumping of an NCP/PEP or EP causes two load modules to be sent to the resource. All activity at the resource ceases, and in order to run again the resource must be activated again from scratch. The DUMPRES command causes the entire storage of the resource to be written in the output dataset.

AUTOMATIC DUMPING AND RESTARTING OF AN NCP/PEP OR EP

If an error condition occurs in the resource, the TSS system will go to the appropriate region of the TSS*****.SYSRCS dataset and check the AUTODMP parameter. If AUTODMP=Y is specified, the TSS system must find the DMPPH1, DMPPH2, and DUMPDS parameters in the region. If all are not present, or if AUTODMP=N or if there is no AUTODMP parameter, the TSS system will bypass the automatic dumping of the resource and process the automatic restart parameter. If RESTART=Y is present in the region, the TSS system will attempt to reload the resource by going through the entire load process.

TRACE OF AN NCP/PEP OR EP LINE

The tracing of the data transmissions of a particular resource line will be initiated by the TRACE command. The data will be sent to the TSS system and then recorded in a dataset for later processing.

SPECIAL COMMAND FACILITIES FOR SYSTEM MONITORS

The system programmer with the privilege class E (system monitor) can reserve unit record equipment for nonconversational tasks (via the SECURE command) and symbolically refer to specific devices (via the DDEF command or macro instruction). In addition, he can use extended PRINT command features, which enable him to have a data set -- previously recorded in the American National Standard Code for Information Interchange, ANSI X3.4-1968, referred to herein as ASCII -- read in from tape and printed out. This lets the TSS system programmer verify an ASCII tape that could have been produced using some other system.

RESERVING I/O DEVICES FOR A NONCONVERSATIONAL TASK

When reserving I/O devices for a nonconversational task, a programmer with privilege class E can designate unit-record equipment in the operand field of the SECURE command. In addition to the options:

(TA=number of devices[,type of device])

(DA=number of devices[,type of device])

shown in the description of the SECURE command in Command System User's Guide, you may ask for one or more printers, card punches, or card readers by specifying one or more of the following operands when using MSAM or IOREQ:

(PR=number of devices)

(PC=number of devices)

(RD=number of devices)

where the number associated with PR indicates the number of high-speed printers you require, the number associated with PC indicates the number of card punches, and the number associated with RD indicates the number of card readers. The number of devices must be specified as a one- or two-digit decimal number.

If you want to reserve two printers and a punch, for example, in addition to three tape units, all for nine-track 800-bpi tape, and one 3330 disk drive for a nonconversational task, you might write:

```
SECURE (PR=2),(TA=3,9D2),(DA=1,3330),(PC=1)
```

Note that the operands need not be written in any particular order.

Because SECURE is defined to reserve "any available device of the specified type", it must not be used when the task requires some particular device (by symbolic device address) of that type.

DESIGNATING I/O EQUIPMENT

When you have been joined with privilege class E, you have several options in the operand field of the DDEF command (and macro instruction), and DCB macro instruction, that are not shown in Command System User's Guide and Assembler User Macro Instructions. Except for these options, which are described in detail here, the parameters you may use are those shown in an appendix to each of those publications.

Symbolic Device Address

One of the options available to you, as a system programmer with privilege class E, is to designate the I/O device you want to use by its symbolic address. This can be accomplished by entering

,UNIT=sda

in the operand field of the DDEF command or macro instruction, where sda is a one-to-four-hexadecimal digit symbol (from 1 to 7FFF) assigned at system generation to the I/O unit as its symbolic device address. By choosing this option, you can designate a particular terminal, a particular unit-record device (for MSAM or IOREQ programming), or a particular tape drive or direct access device (for BSAM, IOREQ, QSAM, or VAM programming).

DDEF by SDA is executed without regard to device reservation via either SECURE or RELEASE ...,HOLD. When a specific device is to be used (regardless of type) there should be no attempt to SECURE it, and RELEASE should always specify the SCRATCH option.

VAM Data Set Allocation on Drums

A user with privilege class G (and only such users) can initially allocate a data set to a drum by specifying the appropriate volume IDs in DDEF. The same restriction and requirement apply to expansion allocation on a drum after the initial DDEF has been released. Use of the SPACE HOLD option of DDEF can allocate drum space for data set extension by non-class G users.

Access to an existing drum data set by a non-calss G user, or by a class G user who does not specify the volume ID(s), is controlled by normal catalog access and sharing authorization. Expansion allocation, in these cases, is forced to disk volumes.

PRINTING DATA SETS

Data sets recorded on tape in EBCDIC or ASCII can be printed in several formats by a class-E system programmer using the PRINT command.

The Printing Options

By specifying the tape printing option (TAPOPT) with the PRINT command, the class-E system programmer can:

- Print an ASCII tape in character format.
- Print an ASCII tape in dump format.
- Examine an ASCII tape for any invalid characters, flag them, and print all error records in the dump format.
- Print an EBCDIC tape in dump format.

Extended PRINT Command Facilities

The basic format of the PRINT command, when employing the extended ASCII options, appears below.

Operation	Operands
PRINT	DSNAME=current data set name[,ERASE=ERASE] [,ERROROPT={ACCEPT|SKIP|END}] [,FORM=paper form] [,TAPOPT={AC|AD|AE|ED|EC}]

Note: To enter positionally, see the PRINT command in the Command System User's Guide. Operands for standard PRINT command options, such as STARTNO, ENDNO, PRTSP/EDIT, and STATION, are recognized when TAPOPT=EC, or the default. If included for the ASCII or EBCDIC options, they are ignored.

The processing and output resulting from the TAPOPT options are indicated below:

AC - The data set, recorded in ASCII, is read from tape without translation to EBCDIC and the entire data set is printed in character format. An unprintable character is represented as a period (.).
Sample output: 132 bytes in the format:

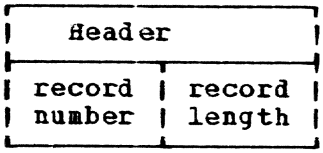
```
........TSS*****.DSNAME...A...
```

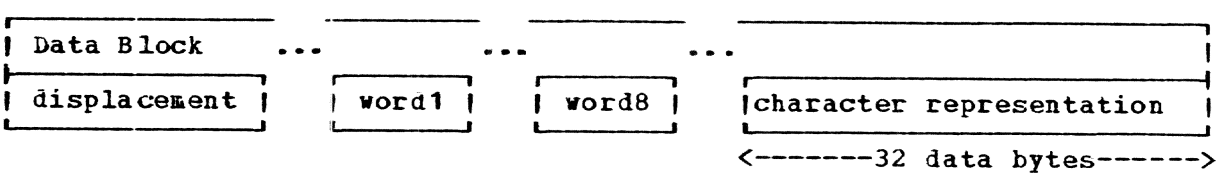
AD,AE,ED - The ASCII dump (AD) and edit (AE) options, and EBCDIC dump option (ED). ASCII records are read from tape without translation to EBCDIC. The entire data set is then printed. The basic print record format for these three options is the same, although certain characters in the printed output will have unique meanings depending on which option is selected. The basic print format for these options appears below, followed by descriptions of the distinguishing print features for each option.

EC - Normal processing. EC is the default if TAPOPT is not specified.

Print Format for AD, AE and ED Options

Each output block consists of a header containing the record number and the record length in decimal. The header record is followed by a data block. Each data record printed consists of the hexadecimal displacement of that data record from the beginning of the block followed by the thirty-two bytes of data recorded in dump format (that is, eight full words, each separated by blanks). The character representation of the data immediately follows.





Distinguishing Features

AD - All unprintable hexadecimal bytes appear as periods. All ASCII error characters (X'1A', or X'80' through X'FF') will appear as

percent signs. Error characters are flagged as such. Since "%" is also a valid print character, each output print record, containing an error byte, is also flagged with *ERROR* appearing in the space between the hexadecimal and character representations; this enables users to distinguish between print records containing valid and invalid percent signs.

Sample output: 64 bytes in hexadecimal, 32 in character

```
DISP   HEX                                      CHARACTER

0000   C1C2C3C4 C1C24040 8080C1C2*ERROR*  ABCDAB %%AB

0020   6C68C1C1 C1C1C1C1                       ...,%.AAAAAA
```

AE - Only error records are printed. The character representation of the hexadecimal data depicts valid ASCII characters as periods (.), a substitute character (X'1A') as an S, and an invalid ASCII character (X'80' through X'FF') as an I. If no invalid characters

are found on the tape, the output consists of an appropriate message. Only ASCII records containing error bytes are printed.

Sample output:

```
0020 C1C1C1C1 1AC180C1  etc.                    ....S.I.  etc.
```

ED - This option does not include any special features.

Sample output:

```
0020 C1C2C3C4 C1C23040 1A80C1C2                 ABCDAB...AB
```

SPECIAL MACRO INSTUCTION FACILITIES FOR SYSTEM MONITORS

MACRO INSTRUCTIONS FOR MSAM

There are four macro instructions that you may use in your MSAM programs. SETUR enables you to specify the unit-record configuration you desire for on-line printers and punches. GET and PUT provide access to logical records and may be specified in either a move mode or a locate mode. FINISH informs the MSAM routines that a break point has been reached in processing a data set.

Interruption Entry Handling

For each of the MSAM macro instructions (SETUR, GET, PUT, and FINISH), a return code of 4 indicates that the operation has not yet been completed. In each case, the macro instruction should be reissued, until a return code other than 4 is received. Before reissuing the macro instruction, however, you should test DCBICB and, if it is nonzero invoke the interruption inquiry routine by issuing the INTINQ macro instruction (described in Assembler User Macro Instructions) to determine whether an asynchronous interruption is pending. If so, you should give control to the appropriate interrupt-handling routine and defer reissuing the MSAM macro instruction until control is returned to your program.

DESIGNATING DEVICES FOR MSAM

In addition to the symbolic device address, three codes may be used with the UNIT operand of the DDEF command and macro instruction when using MSAM. You may write:

```
UNIT={sda|PC|PR|RD}
```

where sda is the symbolic device address of the desired unit record device, PC is a card punch, PR is a printer, and RD is a card reader. If you use the multiple sequential access method, one of these options must be specified.

SECTION 5: DEFINING SYSTEM MACRO INSTRUCTIONS

This section deals with the process of defining macro instructions, concentrating on precautions you should observe and limitations imposed by the various types of macro instructions. It assumes familiarity with the rules for writing macro definitions contained in Assembler Language.

CONVENTIONAL TYPES OF MACRO DEFINITION

In TSS, there are two preferred conventions for defining macro instructions. These conventions define rules for writing the R-type macro instruction, and the standard-, L-, and E-forms of the S-type macro instruction. In describing macro instructions for the user, each macro instruction is designated an R-type, an S-type, or, if one of these conventions was not used, as an O-type (other type).

R-TYPE MACRO DEFINITION

An R-type macro definition can be written when all parameters can be contained in the two parameter registers, 0 and 1. The R-type definition does not generate a parameter list but may generate constants or addresses. You are limited in the choice of operand forms you may allow the user. These forms and coding considerations are described below.

The proper use of an R-type macro is to pass: one or two single-word values (or 4-byte strings); a double word value (or 8-byte string); a control block address and a flag word; or logically similar information. A collection of data organized into a block solely for the use of a given macro, which has no significance beyond the macro or outside the using and called modules, is not a good subject for an R-type macro.

RX Address (formerly known as "explicit" or "implied" address)

This is an address which the user specifies as if he were specifying the second operand of an RX-type assembly language instruction such as L or LA.

This form of address gives the user maximum flexibility in specifying a storage field. It permits addressing by name, addressing by base plus displacement, indexing, and by the effective equivalent of register notation as in: 0(,register). The user must remember to cover with a base register the symbolic addresses that may be written for this form of operand. A portion of the coding of a macro definition called STOR is shown in the box below. &AREA may be written as an RX address. Notice the preferred use of the LA instruction to provide an overriding base register for the STM instruction. You should not write:

```
STM &REGS(1),&REGS(2),&AREA
```

The RX address form permits the coding of indexed addresses. But the STM instruction does not allow for indexing. So, in the example, you would have used the operand &AREA in the LA instruction, which is indexable.

```
&NAME   STOR   &AREA,&REGS
          •
          •
          •
&NAME   LA     14,&AREA
        STM    &REGS(1),&REGS(2),0(14)
```

Number

If you designate an operand to be specified as a number (assumed to be a decimal integer unless you tell the user otherwise) several possibilities must be considered.

If you limit the user to an integer less than 4096, which is not a preferred method, you may write:

```
LA 1, &INT
```

If you allow the user to exceed 4095, which is the preferred method regardless of application you must first test the magnitude of the operand and, in the cases in which it does exceed this value, write:

```
L 1,=F'&INT'
```

The F-type literal is chosen, rather than the A-type, for invariant data, to avoid organizing the literal in the user's first declared PSECT.

You may also choose the number form for an operand that is not a parameter but which serves to indicate the proper path through the macro definition,or the number of iterations of code or data generation to be done by the macro. This type of operand should be treated in conditional assembly instructions.

Absolute Expression

If the value of an operand in the form of an absolute expression is less than 4096, you may use the LA instruction. If this value is greater than 4095, the parameter can be made a literal, which is preferred (as a full word) regardless of the range allowed. For example:

```
L  1,=F'&INT'
```

The absolute expression form of operand may also be used (in exceptional cases) as a path indicator to be used by conditional assembly instructions.

Code

A code value should be enclosed in apostrophes. Some macro instructions offer a code or some other form of operand as alternate choices for designating an operand. In these cases, it would not be possible to distinguish between the alternates without some kind of test. The simplest way to handle this possibility is to require the use of delimiting apostrophes and write your macro definition to test the operand for a leading apostrophe.

If the code value is to be passed in a register as a parameter, restrict it to four characters; if two parameter registers can be used, restrict it to eight. This is a preferred method only for short options related to a single data value or control block address.

You may choose a code to indicate the path to be taken through the macro expansion or to be passed as a parameter in some form other than a character string. In the latter case, you must provide a translation algorithm through the use of conditional assembly instructions.

Character String

You should avoid this operand form in an R-type macro instruction, but might choose to pass a character string parameter in one or a pair of registers. If you choose to do so, be sure to limit the size of the string to conform with the amount of available register space.

You may use a character self-defining term as the displacement field of an LA instruction if the string consists of one character. If the string is longer than one character, your macro definition must employ an L or LM instruction to load a literal.

Symbol

You may specify this operand form if you want to force the writer of the macro instruction to specify a character string that conforms to assembler language conventions.

You may also permit the writer to provide a symbolic name for the first executable instruction in the expansion. If so, be sure to provide for the inclusion of the name in each model statement that may generate the first executable instruction.

Linkage

Many routines called by macro instructions are privileged. If the module issuing the macro instruction is privileged, the macro instruction must generate a type-1 linkage to another privileged module; if the issuing module is nonprivileged, a type-2 linkage must be generated. If a macro instruction may be issued by either type of module, your macro definition must test for the type of linkage you desire to assemble. It can do this by checking the global symbol &CHDCLS that is initialized by the DCLASS macro instruction.

The DCLASS macro instruction is used to tell the assembler which type of linkage you want assembled for macro instructions. If the DCLASS macro instruction specifies USER class or is omitted, &CHDCLS is given a value of 0; if PRIVILEGED is specified, &CHDCLS is given a value of 1.

Some macro instructions generate only type-1 linkages regardless of the issuing module's privilege class. If you write one of these fence-sitters, be sure its entry point name begins with SYS. These characters are used to generate a type-1 linkage.

Finally, some macro definitions generate code without reference to parameters. That is, the same code is generated every time the macro appears in a source program.

EXAMPLE: Here is an example of a typical R-type macro instruction and its associated macro definition, illustrating some of the points just made. Your macro description would be:

```
┌──────────┬─────────┬──────────────────────────────────────────┐
│Name      │Operation│Operand                                   │
├──────────┼─────────┼──────────────────────────────────────────┤
│[symbol]  │RTYPE    │location,length                           │
└──────────┴─────────┴──────────────────────────────────────────┘
```

where location can be specified as an RX address and length must be specified as an absolute expression; your macro definition might look like this:

```
(1)          MACRO                                    HEADER STATEMENT
(2) &NAME    RTYPE     &LOC,&LEN                      PROTOTYPE STATEMENT
(3)          AIF       (T'&LOC EQ 'O').E1             IF 1ST OPERAND IS
                                                      MISSING
(4) *                                                 GENERATE AN ERROR
                                                      STATEMENT
(5) &NAME    LA        1,&LOC                         FIRST GENERATED
                                                      STATEMENT
(6)          AGO       .OP2
(7) .OP2     AIF       (T'&LEN EQ 'O').E2             IF 2ND OPERAND IS
                                                      MISSING
(8)          LCLA      &A                             INITIALIZE SETA
                                                      SYMBOL
(9) &A       SETA      &LEN                           SET VALUE OF SETA
                                                      SYMBOL
(10)         L         0,=F'&A'
(11) .LINK   CHDINNRA  ,,(CZCXYZ),X'FF'
(12)         MEXIT                                    TERMINATE PROCESSING
(13) .E1     ANOP                                     1ST OPERAND MISSING
(14) .E2     ANOP                                     2ND OPERAND MISSING
(15)         MEND                                     TRAILER STATEMENT
```

In this example, line 3 tests for the presence of a first operand and, if it is missing, branches to an ANOP statement in line 13. In practice you would want to place some error processing code at this point. Error processing and the CHDERMAC macro instruction are discussed later.

Line 5 is the model statement which generates the first executable instruction for RX address notation and would also generate the name assigned to the macro instruction.

The second operand is processed in the preferred way.

Finally, line 11 generates the linkage by means of the CHDINNRA inner macro instruction. The third operand, (CZCXYZ), represents the type-1 linkage entry point and the fourth operand represents the ENTER code for type-2 linkage. CHDINNRA determines which type linkage to use.

S-TYPE MACRO DEFINITION

You should employ an S-type macro definition when you wish to generate a parameter list in storage because the parameters cannot be contained in two registers. An S-type macro definition provides the user a choice of three forms of macro expansion: standard, L-form, and E-form.

The standard form, indicated by the omission of the MF=operand, should generate separate E- and L-forms but may directly generate the parameter list and the required linkage to the called routine.

The L-form, indicated by MF=L, generates only a parameter list; it does not generate any executable code. For this reason, the RX address operand form is not allowed in the L-form.

The E-form, indicated by MF=(E,list) where "list" specifies the address of the parameter list as either a relocatable expression or as in RX address notation, generates the proper linkage and may also alter an existing parameter list.

This convention permits the programmer using your macro instruction to conserve space in storage by generating a parameter list by means of the L-form and then altering the same list, in several subsequent calls, by means of the E-form. The L-form is generally intended to be modified by each E-form that uses it.

The placement of the parameter list may be indirectly controlled by the user of your macro instruction, and he should be advised about these precautions:

1. The S-type macro instruction conventionally places the parameter list in the first declared PSECT of the assembled module.

2. If this PSECT is declared by a macro instruction, then that instruction must appear in the user's program before any macro instructions that refer to the list.

3. If rule 1 or 2 is violated, or if no PSECT exists at all, the standard form S-type macro instruction must place the parameter list in line with the code it generates and insert a branch around the list.

4. L-form macro instructions always generate the parameter list in line. Therefore, if the user is writing reenterable code, he will want the parameter list generated in the area occupied by his working storage, presumably his PSECT. This is usually done for him by the standard form S-type. The L-form should only be used in the PSECT.

STANDARD-FORM S-TYPE MACRO DEFINITION

As in the case of R-type macro definitions, the operand forms you allow the user dictate certain steps in your macro definition. Here are standards to observe.

Relocatable Expression

This operand form may be used as the argument of an A-type address constant either in a DC statement or in a literal.

Number and Absolute Expression

If the operand specified by one of these operand forms is an actual numeric value, it is only necessary to generate an F- or A-type address constant, keeping in mind any size constraints that might necessitate the use of length modifiers. Full-word F or A constant is preferred.

Code

If a code is to be passed as a parameter or is to be translated to a value to be passed as a parameter, you must pass it in the parameter

list and not, as in the case of the R-type macro instruction, in a register. Since the code may only be one term, you may use any type of constant to generate the parameter in the list. If the code is a character string that includes apostrophes, you must pass it as a character constant and adhere to the rules for writing such constants. Notice also that the TSS assembler reduces all double apostrophes and double ampersands to single apostrophes and ampersands.

Again, you may choose to use the code as a path indicator. If you wish to pass a variable-length parameter list, you might use a code to indicate the length of the list being passed.

Character String and Text

These types of operand forms may be used in two ways. You can pass the operands to the called routines as character strings in the parameter list; you can generate the character strings and then enter a pointer to them in the parameter list. The latter method is preferred, with the variable length of the string or text generated as a full word immediately preceding the string (no separate field or pointer for length). Since the parameter list produced by the S-type macro instruction normally is a list of pointers,you will, with few exceptions, use this form of operand in character constants or character literals.

You are responsible for verifying the presence of a leading apostrophe in a text operand and for providing error processing if it is missing. The assembler program checks for the terminal apostrophe.

Two methods for checking the length of a character string are available. As you can see in Figure 17, you may test for either the K or the L attributes. The reason for subtracting 2 from the count of &TEXT before placing it in the SETA cell is that the assembler includes the delimiting apostrophes in the count. If you want to find the length of the character string, bear in mind that delimiting apostrophes will have been stripped and double apostrophes and ampersands will have been reduced. Thus, had the programmer written the operand &TEXT as:

```
'USE THIS SYMBOL &&'
```

you would find its K attribute to be 20 (including terminal apostrophes). Statement 3 of the example would yield the value 18 in the SETA cell. Statement 5 would yield a value of 17 in the SETA cell since statement 4 would have generated &TEXT stripped of its terminal apostrophes and one of the two ampersands.

```
(1)              MACRO
(2)              MACX      &TEXT
                   .
                   .
                   .
(3)      &A1     SETA      K'&TEXT-2
                   .
                   .
                   .
(4)      CHDXX   DC        C'&TEXT'
                   .
                   .
                   .
(5)      &A1     SETA      L'CHDXX
                   .
                   .
                   .
                 MEND
```

Figure 17. Determining the length of a character string

Symbol

You may use this operand form in several ways: in the name field of a generated statement, as a character string to be passed as a parameter, or as an entry point or module name to be used as the argument of an address constant, usually R-type or V-type.

L-FORM S-TYPE MACRO DEFINITION

RX addressing is not allowed in the L-form of the S-type macro instruction. The L-form is used to generate a parameter list only. Since RX addressing requires the generation of executable code, it cannot be used.

EXAMPLE A: Coding the L-form part of an S-type macro definition

```
 (1)          MACRO                                        HEADER STATEMENT
 (2)  &NAME   STYPE   &LENLOC,&PROC,&SYM,&SYMLEN,          PROTOTYPE
                        &MF=I
                .
                .
                .
              AIF     ('&MF' EQ 'L').LFORM                 LFORM OF MACRO?
                .
                .
                .
 (3)  .LFORM  AIF     ('&NAME' EQ '').E1                   IS NAME FIELD OK
 (4)          AIF     (K'&LENLOC EQ 0).OMIT1               IS FIRST FIELD OK
 (5)  &NAME   DC      A(&LENLOC)                           ENTER FIRST OPERAND
 (6)  .SYM    AIF     ('&SYM' EQ '').E2                    3RD OPERAND OK
 (7)          DC      CL8'&SYM'                            ENTER 3RD OPERAND
 (8)          AIF     (K'&SYMLEN EQ 0).E4                  4TH OPERAND OK
 (9)          DC      AL1(&SYMLEN)                         ENTER 4TH OPERAND
(10)          LCLB    &B                                   ESTABLISH SETB
(11)  .PROC   AIF     (K'&PROC EQ 0).OMIT3                 IS 2ND OPERAND PRESENT
(12)          AIF     ('&PROC' NE 'E' AND
                       '&PROC' NE 'P').E3                  IS 2ND OPERAND VALID
(13)  &B      SETB    ('&PROC' EQ 'F')                     SET CODE
(14)  .OMIT3  DC      AL1(&B)                              DEFAULT 2ND OPERAND
(15)          MEXIT
(16)  .OMIT1  ANOP                                         DEFAULT 1ST OPERAND
```

```
(17) &NAME  DC      A(0)                          RESUME PROCESSING
(18)        AGO     .SYM
(19) .E1    ANOP
(20) .E2    ANOP
(21) .E3    ANOP
(22) .E4    ANOP
              .
              .
              .
(23)        MEND
```

Example A illustrates this limitation. If you intend to permit the user of your macro instruction to use the L-form, you may wish to emphasize the operand form limitation in your macro description. All other operand forms allowed in the standard form are also allowed in the L-form. Although operands in L-form may be used as path indicators, they are generally used as the arguments of DC statements or are translated to values that are used as arguments.

Since the user controls the placement of the parameter list, do not include a statement to generate a control section; specifically, don't attempt to locate the parameter list in the PSECT.

Figure 18 shows how the S-type macro definition in Example A would be presented to the user. Notice that the name field in the L-form is mandatory. This is a good general rule to follow because most users generate the parameter list and later modify it with an E-form. The name assigned the L-form is a good way to identify the parameter list for later modification.

The coding shown in Example A would generate the parameter list shown in Figure 19. Statements 3, 4, 6, 8, 11, and 12 test for the existence and the validity of each parameter. Statements 5, 7, 9, 14, and 17 generate the parameter list. Notice that lines 3 and 6 use a null test to verify the presence of operands.

Standard form:

Name	Operation	Operand
[symbol]	STYPE	length location,procedure,symbol,symbol length

L-form:

Name	Operation	Operand
symbol	STYPE	[length location],[procedure],symbol,symbol length, MF=L

Figure 18. Standard and L-form S-type macro description

Defaults have been provided in lines 14 and 16. If the second operand is omitted, line 11 branches to line 14 which uses &B as the argument of the address constant. Since &B was initialized to zero by the LCLB instruction, line 14 assumes zero (indicating P).

In your definition, lines 19 through 22 would be followed by error processing.

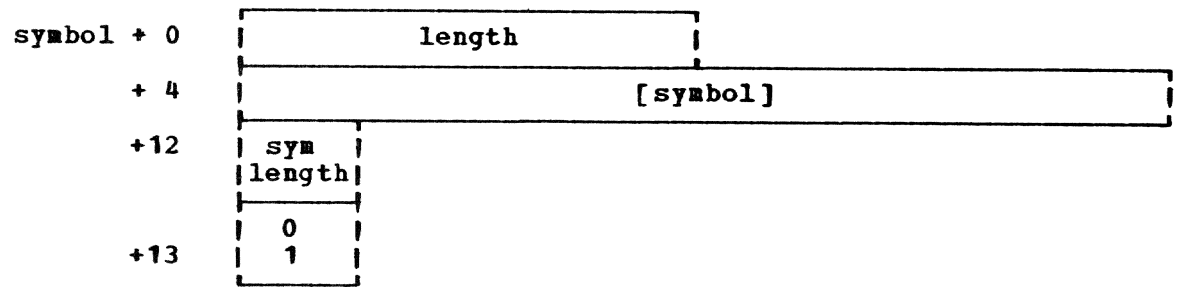


Figure 19. Parameter list generated by L-form

E-FORM S-TYPE MACRO DEFINITIONS

The E-form macro instruction may modify a parameter list and may generate the linkage to the called routine. Since this requires the generation of executable instructions, some changes must be made in the operand forms.

RX Address (formerly known as "explicit" or "implied" address)

This form of operand must be substituted wherever relocatable expression is allowed in the standard form. The use of these forms will difffer from their use in the standard form in that you use them to compute the effective address and then store that address in the parameter list. You should tell the user that he must provide a base register.

By convention, general registers 14 and 15 are used as working registers in the macro definition, because the linkage you generate destroys their original contents anyway.

Number and Absolute Expression

Operands specified in these forms are treated much the same as in the R-type. A Load (preferred) or Load Address instruction is used to load register 14 with the operand value. The choice of instructions again depends on the magnitude of the operand value.

Because RX address notation must be allowed in the MF= operand, you must include a test for it and provide for loading register 1 with the address specified.

Code and Symbol

Operands expressed in these forms may be used as path indicators, and the symbol form may be used to name generated instructions. Although these operands are permitted in the E-form, you will seldom find them useful.

Linkage

When your macro definition is generating the linkage to the called routine, you should generate the entry point in a V-type address constant literal. Not only is this a convenient method, but the assembler program will place this constant in the proper control section -- a PSECT, if one exists.

When generating a type-1 linkage, your E-form macro definition must generate both V-constant and R-constant literals. You can use the inner macro instruction CHDINNRA, which is discussed later, for this purpose.

EXAMPLE B: Coding the E-form part of an S-type macro definition

```
(1)          MACRO                                          HEADER STATEMENT
(2) &NAME    STYPE     &LENLOC,&PROC,&MF=I                  PROTOTYPE
               .
               .
               .
             AIF       ('&MF(1)' EQ 'E').EFORM              EFORM OF MACRO?
               .
               .
               .
(3) .EFORM   ANOP                                           ENTRY POINT
(4) &NAME    DS        0H                                   ALIGNMENT
(5)          CHDINNRA  &MF(2)                               LIST ADDRESS IN
                                                            REGISTER 1
(6)          AIF       (K'&LENLOC EQ 0).PROC                IS THERE A 1ST
                                                            OPERAND
(7)          LA        14,&LENLOC
(8)          ST        14,0(0,1)                            1ST OPERAND TO LIST
(9) .PROC    AIF       (K'&PROC EQ 0).LINK                  IS THERE A 2ND
                                                            OPERAND
(10)         AIF       ('&PROC' NE 'F' AND '&PROC' NE 'P').E1
     *                                                      IS IT VALID
(11)         LCLB      &B                                   ESTABLISH SETB CELL
(12) &B      SETB      ('&PROC' EQ 'F')                     SET SETB CELL
(13)         LA        14,&B
(14)         STC       14,13(0,1)                           STORE CODE IN PARA
                                                            LIST
     *
(15) .LINK   CHDINNRA  ,,(CZCXYZ),X'FF'                     GENERATE LINKAGE
(16)         MEXIT
(17) .E1     ANOP
(18)         MEXIT
```

Figure 20 demonstrates the E-form of the macro instruction described in Figure 18 and the parameter list shown in Figure 19. The coding for a typical E-form S-type macro definition is shown in Example B.

E-form:

Name	Operation	Operand
[symbol]	STYPE	[length location][,procedure],MF=(E,list)

Figure 20. E-form S-type macro description

Note, in Example B, that the only error test is for an invalid second operand. All parameters may be omitted if no change is desired in that field of the parameter list.

MODIFIED R-TYPE MACRO DEFINITIONS

You may choose (in exceptional cases) to pass parameters in registers other than 0 and 1; this makes the definition a modified R-type. If your macro instruction links to the called routine by means of an SVC, you may pass parameters in registers 14 and 15.

No change in operand forms occurs between R-type and modified R-type macro instructions.

MODIFIED S-TYPE MACRO DEFINITIONS

An S-type macro instruction may have no standard form and an E-form that does not generate any linkage. These are modified S-type macro instructions and they serve only to generate and to alter a parameter list.

Another type of modified S-type macro instruction is the type that has only a standard form; it does not have an L-form or an E-form.

TECHNIQUES USED IN WRITING MACRO DEFINITIONS

REGISTER NOTATION

Special register notation should be specified when you wish to allow the user to load parameters (not addresses) into registers before execution of the macro instruction. By convention, these registers are restricted to 0 and 1 for standard R-type macro instructions. Registers 14 and 15 may also be used if the R-type macro instruction is of the modified type. The required method of linkage may place additional restrictions on the use of registers 14 and 15.

Register notation and registers for RX-Address use are limited to registers 2 through 12 in order to avoid the loss of parameters. Assume that you want to pass parameter P1 in register 0 and parameter P2 in register 1. Without this restriction on register usage, the user might issue the HAVOC macro instruction like this:

```
        L    0,P2
        L    1,P1
HAVOC   (0),(1)
```

Your macro expansion would then do this:

```
        LR   0,1      THIS IS PARAMETER P1
        LR   1,0      THIS IS NOT PARAMETER P2
```

If you must use registers other than the conventional ones for working registers, and do not save and restore them, be sure to warn the user.

PACKING PARAMETERS

If you wish to pass two related short parameters or several flags in one register, you must pack the parameters. Figure 21 shows the methods for packing two parameters, each a half word long, into register 1.

	&OPA is register notation		&OPA is an absolute expression	
&OPB is register notation	LR	1,&OPA(1)	LCLA	&A
	SLL	1,16	&A SETA	&OPA*65536
	OR	1,&OPB(1)	L	1,=F'&A'
			OR	1,&OPB(1)
&OPB is an absolute expression	LR	1,&OPA(1)	LCLA	&A
	SLL	1,16	&A SETA	&OPA*65536
	LCLA	&A	L	1,=F'&A'
	&A SETA	&OPB	&A SETA	&OPB
	O	1,=F'&A'	O	1,=F'&A'

Figure 21. Packing two halfword parameters into register 1

Similar techniques can be used for other cases. Here are two examples:

EXAMPLE A:

Parameter P1 - three bytes left aligned
Parameter P2 - one byte right aligned
Both parameters given in register notation

```
          Procedure
          LCLA  &A
&A        SETA  &P1*256
          L     1,=F'&A'
&A        SETA  &P2
          O     1,=F'&A'
```

EXAMPLE B:

Parameter P1 - one byte left aligned
Parameter P2 - three bytes right aligned
Both parameters given as absolute expressions.

```
          Procedure
          LCLA  &A
&A        SETA  &P1
          L     0,=F'&A'
&A        SETA  &P2*256
          L     1,=F'&A'
          SRDL  0,8
```

DEFINING INNER MACRO INSTRUCTIONS

The inner macro instruction CHDINNRA has been used in previous examples. You will find this type of instruction not only convenient but also economical in terms of lines of code written, lines of code generated, and time expended in assembly.

You needn't write as much code in your outer macro definition, since the inner macro instruction will supply it for you. You may also reduce the number of generated statements by using conditional calls. You might, for example, write:

```
          AIF   ('&OP' LE 5).NOINR
          INNERMAC
           .
           .
           .
.NOINR    ANOP
           .
           .
           .
```

The inner macro instruction would only be called if &OP were greater than 5. As you can see, the use of the inner macro instruction is somewhat analogous to the use of a subroutine.

Basically, the reasons for using inner macro instructions are the same as those for using subroutines. If time and space will be saved, define and use an inner macro instruction.

Don't nest more than three levels of macro definition. This technique will keep the definition clean and intelligible.

Don't define an inner macro instruction for use with only one outer macro instruction; use a conditional assembly subroutine instead. Assume that you want to conditionally enter a subroutine from points A, B, and C. You must provide a means by which the subroutine can return to the correct point after each of the three calls. You can do this by establishing a SETC cell and altering its contents prior to each conditional branch. Example C illustrates this technique.

EXAMPLE C: Branching to and returning from a conditional subroutine

```
          MACRO
           .
           .
           .
          LCLC   &RTRN
           .
           .
           .
&RTRN     SETC   '.RTRN1'
          AIF    (&OP1 GT 10).SR
.RTRN1    ANOP
           .
           .
           .
&RTRN     SETC   '.RTRN2'
          AIF    (K'&OP2 EQ 0).SR
.RTRN2    ANOP
           .
           .
           .
&RTRN     SETC   '.RTRN3'
          AIF    (L'&OP2 LT 1).SR
.RTRN3    ANOP
           .
           .
           .
          MEXIT
.SR       ANOP
           .
           .
           .
          AGO    &RTRN                    (END OF SUBROUTINE)
           .
           .
           .
          MEND
```

NAMING THE FIRST EXECUTABLE INSTRUCTION

It is possible to put the name field of the macro prototype statement on the first executable instruction even when there is more than one model statement that may be generated as the first executable instruction. It is preferred, however, to use the name in the name field of a non-executable assembler statement (for example, &NAME DS 0H or &NAME EQU *).

SETTING THE SIGN BIT

If you define a macro instruction in which the user may specify the sign of operand two by the presence (negative) or the absence (positive) of operand one, you may find it difficult to properly set the sign bit. Examples D and E illustrate two techniques you might find helpful.

After establishing SETA and SETB cells, line 5 places the proper bit value in the SETB cell, based on the presence or the absence of OP1. Line 6 then generates the parameter in storage using the value in the SETB cell for the sign. Notice that the length modifiers in line 6 specify the length in bits. You must use one line for the DC statement. If you use two lines, the second line will be aligned on a byte boundary and the sign bit will be lost.

Example E illustrates another technique; it uses the same MACRO, PROTOTYPE, and LCLA statements from Example D. In Example E, line 7 tests for the absence of OP1. If it is absent, the sign is to be positive and lines 15 and 16 generate a positive value in register 1.

If the sign is to be negative and OP2 is zero, line 12 generates a negative zero in register 1.

If OP2 is to be a negative number other than zero, line 9 computes the two's complement of OP2 and places it in the SETA cell. Line 10 loads register 1 with the negative SETA symbol. The assembler will convert the value in &A to its negative two's complement. Since the value in &A is already the two's complement of OP2, line 10 will load the absolute value of OP2 with the sign bit on.

EXAMPLE D:

```
(1)             MACRO
(2)   &NAME     MACEX   &OP1,&OP2
(3)             LCLA    &A
(4)             LCLB    &B
(5)   &B        SETB    (K'&OP1 NE 0)
                 .
                 .
                 .
(6)             DC      AL.1(&B),AL.31(&OP2)
                 .
                 .
                 .
```

EXAMPLE E:

```
 (7)            AIF     (K'&OP1 EQ 0).OMIT
 (8)            AIF     (&OP2 EQ 0).ZERO
 (9)   &A       SETA    X'7FFFFFFF'-(&OP2-1)
(10)            L       1,=F'-&A'
(11)            AGO     .DONE
(12)  .ZERO     L       1,=X'80000000'
(13)            AGO     .DONE
(14)  .OMIT     ANOP
(15)  &A        SETA    &OP2
(16)            L       1,=F'&A'
(17)  .DONE     ANOP
                 .
                 .
                 .
```

PROCESSING A SINGLE APOSTROPHE

You must exercise caution in the treatment of operands that may validly contain single apostrophes. If, for example, a single apostrophe is found in a character relation in an AIF instruction, it will produce invalid syntax.

There is a way to test for single apostrophes without violating syntax rules. You might write

```
AIF ('&OPND'(1,1)'&OPND'(1,1) EQ '''').TEXT
```

This use of substring notation concatenates the operand field you want to test with itself, thereby generating a pair of the tested character. Thus, if the character tested is an apostrophe, paired apostrophes will be produced and no violation of the rules of syntax will result.

It's worth noting here that there are three methods available to you to test for the presence of an operand

```
1.  AIF (K'&OPERAND EQ 0).OMIT
2.  AIF (T'&OPERAND EQ 'O').OMIT
3.  AIF ('&OPERAND' EQ '').OMIT
```

Method 1 tests for a count of zero, method 2 tests for a type of "omitted", and method 3 tests for a null character string. You should not use method 3 if it is possible for a single apostrophe to appear in the operand. A test for the K attribute is your best course.

REFERRING TO THE DCB

If the macro instruction you define must refer to the user's DCB, you should express references to the various fields in terms of symbolic notation. Reference to fields in terms of byte displacement is bad practice; if the DCB is ever redefined, your byte displacements may no longer be correct. The use of symbolic field names requires the user to have previously issued a DCBD macro instruction or the macro instruction currently being defined must issue a DCBD inner macro instruction. Since the DCBD macro instruction uses the global symbol &CHDDCBD to prevent multiple expansions of the DCBD macro instruction in one assembly, there is no reason why DCBD should not be coded as an inner macro instruction to allow symbolic references to the user's DCB. If you want to avoid the error message normally issued for duplicate DCBDs, you can test &CHDDCBD within the macro instruction code before issuing DCBD and branch around it if the value is not equal to 0. (Note, however, that if the user has copied CHADCB and issues DCBD, duplicate symbols will result.)

SIZE LIMITATION

If the operand is not a sublist, it must not contain more than 255 characters. If the operand is a sublist and the only references are to individual members of the sublist, each member may be up to 255 characters long. You are not restricted in the number of operands or in the number of sublist elements.

ADDRESS CONSTANTS

If an R-type address constant refers to a symbol defined in a program that has no PSECT, then the R-value defined is the origin of the control section containing the ENTRY statement whose operand field contains the argument of the R-type constant. Thus, given R(X), where X is defined in an assembly module having no PSECT, the R-value is the origin of the control section containing the statement:

```
        ENTRY X
```

If there is a PSECT, all address constant literals are located in it. If no PSECT exists, the constants are placed in a literal pool.

V-type and R-type constants must have only a single relocatable symbol as an argument. If an operand is to become the argument of such an address constant, you should show the operand form as "symbol."

A symbol X and a V-type constant with an argument of X may both be defined in one assembly module. Unless X is also defined as an entry point to the module, the V-type constant is resolved by searching for a definition of X outside the current module.

Testing the type attribute of a symbol for the value T will only indicate whether it is defined as the operand of an EXTRN statement in the assembly. If a symbol is externally defined as the argument of a V-type or R-type address constant, its attribute will be given as U for undefined. This cannot be considered as a conclusive test, however, since U is also the attribute assigned to symbols internally defined by an EQU statement. The test for a type attribute of T can only be used to indicate that a symbol was defined by means of an EXTRN statement.

If your macro instruction generates an implicit adcon group and may be called from a user-written program, it is not safe to assume that the user has defined entry points with either EXTRN or ENTRY statements. The type of constants you generate should be determined by testing for the T-value of the type attribute. If it is present, you may generate a V-type and R-type constant pair. If it is not present, generate an A-type constant pair. Admittedly, the user may have defined the symbols with ENTRY statements but, since there is no way to test for them, this is your only safe course.

Conventionally, the R-value of the A-type constant is assigned from &SYSPSCT (that is, the first PSECT). If this variable is null, indicating that no PSECT exists, the R-value is assigned (from &SYSECT) the origin of the CSECT from which the macro instruction has been issued.

An exception to this convention occurs in the ADCON macro instruction. Since the user controls the placement of the ADCON macro instruction and probably wishes the adcon group constructed in the same PSECT, the R-value is always assigned the value from &SYSECT. You should use this same technique when you want to let the user declare more than one PSECT and generate the adcon group in a PSECT other than the first.

TERMINAL APOSTROPHE AND SIZE LIMITATION

Assume that a user writes a text operand that is 258 characters long, including terminal apostrophes. After you have tested for the initial apostrophe, you seek to determine the K attribute. Since this attribute is expressed in modulo 256, you would receive a character count of 2: the initial apostrophe and the first character. The terminal apostrophe would be missing and an error message would be generated by the assembler program.

KEYWORD OPERANDS AND STANDARD VALUES

If you write a macro definition and include a keyword operand to which you assign a standard value, the type attribute of the standard value is assigned to the operand if it is completely omitted by the user. If he writes KEYWORD= and follows the equal sign with a blank or a comma, the type attribute of the operand is O for omitted, and the standard value is overridden by the explicitly specified null string.

SUBSTRING NOTATION PROCESSING

If you use substring notation to refer to a subset of characters in a character string, you must first ensure that the characters are present. Assume, for example, that you want to test the first four characters of an operand to see if they specify some specific action to be taken. You should write something like this:

```
        AIF  (K'&OP LT 4).ERROR
        AIF  ('&OP'(1,4) EQ 'REG1').PROC
```

If you don't do this and the user codes a character string of less than four characters, the assembler will produce error messages.

This technique must be employed where register notation is allowed. Since you will use substring notation to test for the opening parenthesis, you must first determine that the operand has been coded. The user may have chosen to omit the operand.

Notice also that you can gain access to a character subset in an element of a sublist by writing something like this:

```
'&OPERAND(2)'(1,1)
```

This refers to the first character of the second element in the sublist &OPERAND.

N ATTRIBUTE USAGE

The N attribute counts the number of operands or the number of elements in a sublist by counting the number of commas and adding one. As a result, the N attribute cannot be used to count the number of non-null operands or non-null elements in a sublist.

N'&SYSLIST HANDLING IN MIXED MODE MACRO INSTRUCTION

Keyword operands are not included in the value of the N attribute of &SYSLIST in mixed mode operands. If there are no positional operands, N'&SYSLIST is zero.

SUBSCRIPTS AND SUBLISTS

If a subscripted reference is made to an operand that is not a sublist, the whole operand is used. Thus, if you write DC A(&OP(15)) and the operand is not a sublist, you generate the operand as the argument of the A-type constant, just as if you had written DC A(&OP).

SETC SYMBOL LENGTH

The maximum length of a SETC symbol is eight characters. As a result, you may not be able to use in its entirety the operand of a SETC statement written as a relocatable expression, absolute expression, or character string. Instead, it is better practice to use the operand in groups of eight, being careful to test for the presence of characters before attempting to use them.

For example:

```
(1)              MACRO
(2)     &NAME    INSTR    &P1
(3)              LCLA     &CT1,&CT2,&CT3
(4)              LCLC     &S(6)

(5)     &CT1     SETA     (K'&P1(2)-2)
(6)     &CT2     SETA     1
(7)     &CT3     SETA     2
(8)     .LOOP    AIF      (&CT1 LE 8).OUT

(9)     &S(&CT2) SETC     '&P1'(&CT3,8)
```

```
(10)    &CT1        SETA      &CT1-8
(11)    &CT2        SETA      &CT2+1
(12)    &CT3        SETA      &CT3+8

(13)                AGO       .LOOP

(14)    .OUT        ANOP
(15)    &S(&CT2)    SETC      '&P1'(&CT3,&CT1)

(16)    &NAME       &P1(1)    &S(1)&S(2)&S(3)&S(4)&S(5)&S(6)

(17)                MEND
```

The INSTR macro is given one parameter that is a two (2) element sublist. The first element is an instruction and the second element is the operand field for that instruction expressed as a character string. The local SETC symbol, &S(6), is a six (6) element array into which we can put the operand field from &P1. By concatenating this SETC array in line 16, we can generate the requested instruction. In this example, &P1(2) is limited to 48 characters.

LOGICAL TERMS IN RELATIONAL EXPRESSIONS

When a relational expression is used in the operand of an AIF or SETB statement, the terms on either side of the relational operator must both be arithmetic expressions or character expressions; neither of the terms can be logical expressions. This is illustrated in the examples below, only some of which are valid. &B(1), &B(2), and &B(3) are SETB variables.

```
valid       AIF     ((&B(1)+&B(2)+&B(3)) NE 0).ON
invalid     AIF     ((&B(1) OR &B(2) OR &B(3)) EQ 0).ON
invalid     AIF     (&B(1) EQ 0).ON
valid       AIF     ((&B(1)+0) EQ 0).ON
valid       AIF     (&B(1)).ON
valid       AIF     (&B(1) OR &B(2) OR &B(3)).ON
```

CONVERTING DECIMAL TO HEXADECIMAL

Sometimes it may be useful to convert an input decimal number to its hexadecimal equivalent. The following example is one way this may be done:

```
            MACRO
&NAME       CONV      &DEC
            LCLC      &STR
            LCLA      &D,&Q,&R
&D          SETA      &DEC
            AIF       (&DNE 0).STRT
&STR        SETC      '0'
            AGO       .OUT
.STRT       AIF       (&D LT 16).END
.LOOP       AIF       (&D LT 16).DONE
&Q          SETA      &D/16
&R          SETA      &D-16*&Q
&STR        SETC      'FEDCBA9876543210'(16-&R,1).'&STR'
&D          SETA      &Q
            AGO       .LOOP
.DONE       ANOP
&STR        SETC      'FEDCBA9876543210'(16-&D,1).'&STR'
            AGO       .OUT
.END        ANOP
&STR        SETC      '123456789ABCDEF'(&D,1)
```

```
.OUT      ANOP
&NAME     DC       XL4'&STR'
          MEND
```

SETTING UP FLAG BITS IN A BYTE

Sometimes it is necessary to generate a byte that contains bit flags that will be meaningful to a program that is going to be called. Here is an example of how to do this using local SETB symbols:

```
          MACRO
&NAME     FLAG     &P1,&P2,&P3,&P4
          LCLA     &FLG
          LCLB     &B(4)

&B(1)     SETB     ('&P1' EQ 'F1')
&B(2)     SETB     ('&P2' EQ 'F2')
&B(3)     SETB     ('&P3' EQ 'F3')
&B(4)     SETB     ('&P4' EQ 'F4')

&FLG      SETA     &B(1)*1 +&B(2)*2 +&B(3)*4 +&B(4)*8

&NAME     DC       AL1(&FLG)   FLAG BYTE
          MEND
```

The four parameters on the FLAG macro can be specified as F1, F2, F3, and F4 respectively. A local SETB symbol is set if any flags are specified. Then, each SETB symbol is multiplied by the decimal equivalent of the hexadecimal number that represents its flag bit within the flag byte (that is, the flag bit for F4 is a X'08' or decimal 8). The results of the four multiplications are added and the result placed in the DC instruction that will generate a byte containing the appropriate flag bits.

GAINING ACCESS TO MACRO LIBRARIES

The macro instructions defined for your use in TSS are divided among several libraries. The first, TSS*****.SYSMAC, contains those macro definitions needed to support nonprivileged user assemblies; the external description of each of these macro instructions can be found in this manual or in Assembler User Macro Instructions. A second, TSS*****.ASMMAC, contains other system macro definitions, such as those for the on-line test system (OLTS), and the system DSECTs. This division permits each installation to move the less frequently used library, TSS*****.ASMMAC, out of public storage (by means of the VT command), freeing the space for other purposes. If this is done, however, the macro library must be renewed on public storage (by means of the TV command) before it can be used. Other macro libraries exist for special system applications and are documented with those applications.

While the macro instructions contained in TSS*****.SYSMAC are always available to you without having first to define the library, this is not true of the macro instructions contained in the other library. To gain access to TSS*****.ASMMAC, you must first issue the following SHARE command (unless your user identification is TSS, in which case the library is already in your catalog):

```
SHARE  DSNAME=dsname1,USERID=TSS,OWNERDS=ASMMAC
```

where dsname1 is any valid data set name you choose to give to the library. You must also issue a SHARE command to gain access to the library index:

```
SHARE  DSNAME=dsname2,USERID=TSS,OWNERDS=ASMNDX
```

where dsname2 is any valid data set name you choose to give to the index.

Although both ASMMAC and ASMNDX are generation data sets, only these two SHARE commands need be issued, since they provide access to all generations. Furthermore, you need only issue these commands the first time you want to use the library; your sharer's status is permanently recorded by the system at that time. However, in each session in which you want to use the library, including the first, you must also issue the following DDEF commands, even if your user identification is TSS:

```
DDEF  DDNAME=ddname1,DSORG=VI,DSNAME=dsname1(x)
DDEF  DDNAME=ddname2,DSORG=VS,DSNAME=dsname2(x)
```

where ddname1 and ddname2 are any valid data definition names of your choosing; dsname1 and dsname2 are the data set names you assigned to the library and index, respectively, when you entered the SHARE commands; and (x) denotes the absolute generation number of the library and index you wish to access. (0) would indicate the latest generation; (-1) would indicate the next most recent generation. Note that the data definition names entered here as ddname1 and ddname2 must be entered as the ddname of the symbolic portion of the supplementary macro library and the ddname of the index portion of the supplementary macro library in the parameter list following the ASM command if you want to assemble macro instructions defined in TSS*****.ASMMAC.

PART II: SYSTEM MACRO INSTRUCTIONS

This part describes macro instructions of special interest to the system programmer. They supplement the macro instructions available to all TSS users, described in Assembler User Macro Instructions.

Some system macro instructions can be issued only by a privilege class E user (system monitor), a privileged system programmer (privilege class D, authority code O), or a nonprivileged system programmer (privilege class D, authority code P). Other system macro instructions have no privilege requirements; they simply make system coding easier.

Section 1 explains how the macro instructions in this book are described.

Section 2 describes the macro instructions, arranged alphabetically.

First, to understand how macro instructions in this book are described, look at Figure 22. This figure may also serve as a quick reference when reading a macro instruction description. You may wish to tab it.

The information which follows supplements Figure 22. A more detailed explanation of operand forms defined by the assembler language is contained in Assembler Language, GC28-2000.

Name Field

In most macro instructions, the symbol in this field becomes the name of the first instruction generated.

Operand Field

The operand field may contain no operands (in which case this is noted below the format illustration), or one or more operands separated by commas.

The user must supply positional operands in the same order as that shown. If a positional operand is omitted and another positional operand is written to the right of the omitted operand, the comma that would have preceded the omitted operand must be retained. For example, assume positional operands A, B, and C. These may be written:

A,B,C	A,,C	A,B	A	,B	,,C

OPERAND FORMS

Absolute Expression

An absolute expression may be an absolute term or any arithmetic combination of absolute terms; an absolute term may be a single absolute symbol or self-defining term. All arithmetic operations are permitted between absolute terms.

In the following examples, ALAN and JAY are relocatable and defined in the same control section; MARK and ERIC are absolute:

```
331
MARK
MARK+ERIC-2
ALAN-JAY
MARK*4-ERIC
```

Relocatable Expression

A relocatable expression is one whose value would change by n if the program in which it appears is relocated n bytes away from its originally assigned area of storage. All relocatable expressions must have a positive value. A relocatable expression may be a single relocatable term. A relocatable expression may contain relocatable terms -- alone or in combination with absolute terms -- under the following conditions:

1. There must be an odd number of relocatable terms.

2. All relocatable terms but one must be paired.

3. The unpaired term must not be directly preceded by a minus sign.

4. A relocatable term must not enter into a multiply or divide operation.

A relocatable expression reduces to a single relocatable value. This value is the value of the odd relocatable term, adjusted by the values represented by the absolute terms or paired relocatable terms, or both, associated with it. The attribute belongs to the odd relocatable term. Complex relocatable expressions are also permitted. Refer to IBM Time Sharing System: Assembler Language, GC28-2000.

In the following examples of relocatable expressions, SAM, JOE, and FRANK are in the same control section and are relocatable; PT is absolute.

```
SAM
SAM-JOE+FRANK
JOE-PT*5
SAM+3
```

Note that SAM-JOE is not relocatable, because the difference between two relocatable addresses is constant.

Register Notation

Register notation is an absolute expression enclosed in parentheses. The absolute expression, when evaluated, must be some value 2 through 12 (unless noted otherwise in the description of the macro instruction), indicating the corresponding register. In these examples of register notation, PAL is absolute:

```
(5)    indicates register 5
(PAL)
(PAL+3)
```

RX Adress (formerly called "explicit" or "implied" address)

An address is written in the same form as an assembler-language operand:

```
a(b,c)
▲ ▲ ▲
| | └──base register
| └────index register
└──────displacement
```

Examples are:

```
2(0,5)
0(2,4)
INITIAL
ALPHAY(4)          ALPHAY is indexed by the value in Register 4.
```

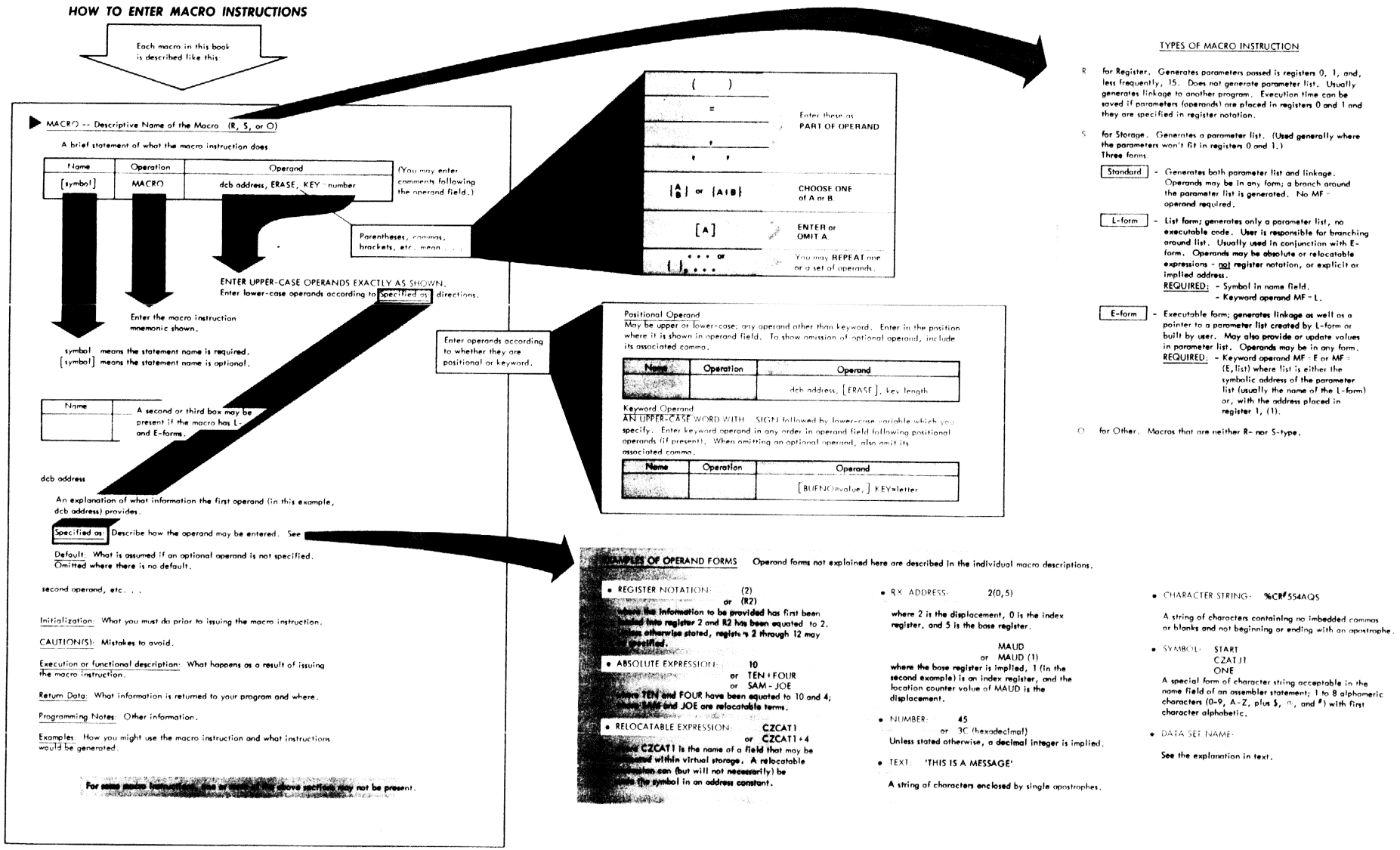


Figure 2. How to enter macro instructions

Symbol

A symbol may be a symbolic address (that is, a single relocatable term), such as the name of an instruction in an assembler-language program, or it may be a character string used for identification, not location (such as the ddname parameter of a DCB macro instruction).

In TSS, the alphabetic characters are the upper case letters A-Z, and $, @, and #. The alphameric characters are the alphabetic characters plus the digits 0-9.

The symbol is written as a string of up to eight alphameric characters, the first of which is alphabetic. Embedded commas and blanks are not permitted. Symbols beginning with the characters CHD may not be used, since symbols beginning with those characters are reserved for system use. Examples of symbols are:

```
DDNAME1
LOOP12
START
#1
```

Character String

A character string may contain any sequence of characters, with the following exceptions: embedded commas or blanks are not permitted. Two apostrophes or two ampersands must be used to represent one apostrophe or one ampersand in the character string. The character string may not be enclosed in apostrophes. For example:

```
CC#SU2GOT@DO&&GO
```

Text

A text operand is written as a string of alphameric characters enclosed in apostrophes. Embedded blanks and special characters are permitted. Two apostrophes or two ampersands must be used to represent one apostrophe or one ampersand in the character string. The text operand may not exceed 255 characters including the enclosing apostrophes. For example:

```
'AREA,PCB,132, ,1256'
```

Data Set Name

This is the name of one data set or a group of data sets. The rules for writing data set names are presented below; the types of names that may be written for each macro instruction are under each macro instruction's description.

Fully qualified name: Uniquely identifies one data set.

1. A stand-alone data set name identifies a data set that is neither a member of a partitioned data set nor a generation of a generation data group. The name of a stand-alone data set is written as a series of symbols separated by periods. For example:

   ```
   DATASET.TRIAL.TEST1
   TERI.ROGER.LAURIE
   A.B.C
   ```

   The rightmost symbol is the data set's simple name (TEST1, LAURIE, and C above); the other symbols are qualifiers. In TSS, for cata-

loging purposes, the maximum number of characters in a data set name including periods, is 35. The maximum number of characters in any single qualifier is 8. The maximum number of one-character qualifiers for a one-character name is 17.

Note: Data set names created under the IBM OS Operating Systems can contain a maximum of 44 characters; if data sets with names greater than 35 characters are to be cataloged in TSS, the user should employ the renaming facility of the CATALOG command to define a suitable TSS name.

2. A partitioned data set and member name identifies a data set that combines individual data sets, called members, into a single data set. The partitioned organization allows the user to refer to either the entire data set or to an individual member of the partitioned data set.

The rules for writing the name of a partitioned data set are the same as for writing those of a stand-alone data set.

The rules for writing a member name vary with each macro instruction that can manipulate members. Sometimes (as in LOAD and DELETE) only the simple member name (a symbol) is written. The full name is not required because the user has indirectly defined the partitioned data set (library) in which the module resides by assuring that the library is on the program library list prior to issuing those commands. The user can write

```
LOAD SORTR
```

if he has previously entered SORTR in a library currently on the program library list.

In other macro instructions (for example, CDS), the user must give the fully qualified member name. This consists of the name of the partitioned data set suffixed by the simple member name in parentheses. For example:

```
HQW(ONETRY)
G.H.AB(H)
```

Here HQW and G.H.AB are partitioned data sets with members ONETRY and H, respectively.

The name of the partitioned data set is written with the same rules as for a stand-alone data set. The parentheses and member name are considered as an appendage to that name. The maximum number of characters in a member name is 8.

3. Generation names identify data sets that are part of a generation data group. These data sets can be referred to absolutely or relatively:

   a. Absolute generation names are written as the name of the generation data group followed by a period and the characters GxxxxVyy, where xxxx is a four-digit, decimal, generation number, and yy is a two-digit, decimal, version number. For example:

```
HURST.LINER.TT.G0001V00
HJ.LA4.WW.G0003V01
HARQ.G0147V03
```

The characters GxxxxVyy are considered a fixed-part of the overall name. The name of the generation data group is a partially qualified name applicable to all generations in the group.

If the generation is a partitioned data set, a member (for example, JOE) within that data set is referred to as follows:

```
A.B.C.GxxxxVyy(JOE)
```

b. Relative generation names are written as the name of the generation data group followed by the appropriate relative generation number enclosed in parentheses:

```
G.D.G(0)
```

The relative generation number of the most recent generation is (0); the generation just prior to that is (-1); the one before that is (-2), etc.; and a new generation to be added is (+1). For example:

```
GOST.UU.L19P(+1)
GOST.UU.L19P(-3)
MRQ.T.L5.SWIM(0)
```

If the generation is a partitioned data set, a member within that data set is referred to as follows:

```
SEAT(-3)(JOE)
```

where JOE is the member in question.

Partially qualified names: Refer to all data sets having the partially qualified name as their common higher-order qualifier.

1. A generation data group name is the name that is common to each generation in the group. Generation data group names are restricted to a maximum of 26 characters including periods.

2. Another partially qualified name can also be used to refer to two or more data sets. For example, the partially qualified name GO.AB14 can be used to refer to both of the following data sets: GO.AB14.A and GO.AB14.B. If these were the only two of a user's data sets with the same higher-order qualifier, GO.AB14, and he wished to erase them both, he could do so by specifying GO.AB14 in the ERASE macro instruction.

ADDEV -- Add Device to Task Symbolic Device List (R)

The ADDEV macro instruction adds an I/O device to your task's symbolic device list.

Name	Operation	Operand
[symbol]	ADDEV	[device]

device
specifies the symbolic device address of the particular I/O device you want added to your task's symbolic device list (TSDL).

Specified as: A number assigned by your installation as the device's symbolic device address in register notation (2 through 12). The number must be loaded into the specified register before issuing the macro instruction.

Default: It is assumed that the issuer has placed the symbolic device address in register 0.

Execution: The resident supervisor adds an entry for the specified symbolic device address to the task's symbolic device list. If the device is already in the task's symbolic device list, the count of the number of times the device has been added is increased by 1; if this count exceeds 15 an error is indicated.

Return Data: The resident supervisor sets the high-order bit of register 0 to 1 if the count of the number of times the device has been added exceeds 15.

Example: Suppose you want to add symbolic device 17 to your symbolic device list. You might write:

```
        LH   2,=Y(17)
ADD     ADDEV  (2)
```

ADDPG -- Add Virtual Storage Pages (R)

The ADDPG macro instruction is used by the system to properly maintain the status of a task's page and segment tables. It adds contiguous pages to a task's virtual storage and creates any necessary page table and external page table entries. A GETMAIN request for virtual storage causes an ADDPG to be issued.

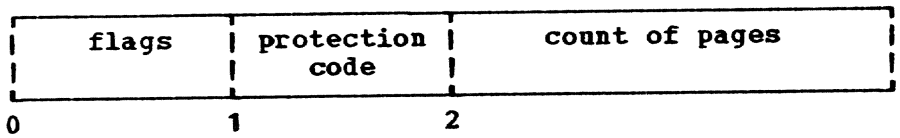
Name	Operation	Operand
	ADDPG	

Note: There are no operands.

Initialization: Before the ADDPG macro instruction is executed, the issuing program must set up the following general registers:

Register(s)	Contents
R0-R1	Reserved segment name (RNAME), if any
R15	Flags, protection code, count of pages requested

The format of R15 is:



where:

Flags	Meaning
X'01'	variable request
X'02'	system program request
X'04'	return code requested
X'40'	RNAME given

Protection Code	Meaning
0	non-privileged read/write pages requested
1	non-privileged read only pages requested
2	privileged pages requested

Execution: New page table entries and, if necessary, segment table entries are constructed. The number of page table entries to be constructed is determined by the page count contained in register 15. The second byte of register 15 is used to determine the setting of the storage keys for all the pages being added.

Return Data: If a return code has been requested, register 15 contains the following codes:

Code	Meaning
0	allocation done
4	request not satisfied

Otherwise an extended program interrupt is enqueued on the task for the 'request not satisfied' return. The virtual memory address of the first page allocated is returned in general register 1.

Programming Note: Page allocation for privileged programs is from the top of virtual memory down. Page allocation for non-privileged programs is from the bottom of virtual memory or the bottom of the RNAME area up. If PACKSEG is SYSGENed, pages are allocated without respect to any boundary alignment. If PACKSEG is not SYSGENed, non-privileged program requests are allocated on 16 page boundaries for requests that are multiples of 16 pages and on 256 page boundaries for 256 page requests.

The systems programmer should use the GETMAIN macro instruction described in Assembler User Macro Instruction, GC28-2004, to allocate virtual memory to a task.

Example: Assume that 100 pages of virtual storage are to be allocated with non-privileged, read only protection keys. The following instructions might be written:

```
.
.
.
SR    0,0                  NO RNAME
SR    1,1                  NO RNAME
LA    15,100               PAGE COUNT
O     15,=A(X'00010000')   PROTECTION CODE
O     15,=A(X'04000000')   RETURN CODE REQUESTED
ADDPG
.
.
.
```

ADSPG -- Add Shared Virtual Storage Pages (R)

The ADSPG macro instruction is used by the system to add shared pages to a task's virtual storage and create any necessary shared page, segment, and auxiliary segment table entries. The ADSPG macro instruction is issued by the system to maintain the status of a task's segment and shared page tables. The system programmer should use the GETSMAIN entry point (CZCGA6) in VMA to obtain shared virtual storage.

Name	Operation	Operand
[symbol]	ADSPG	

Note: There are no operands.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding ADSPG. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding ADSPG must be issued with the PRIVILEGED option. Before the ADSPG macro instruction is executed, the issuing program must set up the following general registers:

Register(s)	Contents
R0	protection code
R1	count of pages requested
R15	zero or connected shared page table number

where:

Protection Code	Meaning
0	non-privileged read/write pages requested
1	non-privileged read only pages requested
2	privileged pages requested

Execution: The number contained in bytes 2 and 3 (the low-order bytes) of register 15 is used to determine if pages are to be added to an existing shared page table or if a new shared page table is to be constructed. If bytes 2 and 3 of register 15 are zero, or if there are not enough pages remaining in the shared page table indicated by bytes 2 and 3, a new shared page table is constructed. Once the shared page table is selected or constructed, a number of page table entries corresponding to the number contained in register 0 is added to it. Storage protection keys are assigned as requested by the code in byte 3 of register 0.

Return Data: The following data are returned in general registers:

Register(s)	Data
R0 (bytes 0-1)	relative page number within shared page table
R0 (bytes 2-3)	shared page table number
R1	address of first page allocated
R15	0 - allocation done 4 - request not satisfied

Programming Note: Allocation of shared pages is controlled by two variables:

1. Connected SPT# given as input
2. PUBSEG specified at SYSGEN

The following rules govern allocation:

a. Allocation of shared segments is from top of virtual memory down.

b. Allocation of pages within shared segments is from the bottom up.

c. Pages are allocated in the SPT# given as input.

d. If "c" fails or if an SPT# was not given, pages are allocated in any existing SPT# if PUBSEG was SYSGENed.

e. If "d" fails or if PUBSEG was not SYSGENed, pages are allocated in a new SPT#.

Example: Suppose two shared non-privileged read only pages are to be added; assume the shared page table number is X'FFF3'. The macro instruction might be written:

```
.
.
.
LA    0,1                      PROTECTION CODE
LA    1,2                      COUNT OF PAGES
L     15,=A(X'0000FFF3')       SHARED PAGE TABLE NUMBER
ADSPG"
.
.
.
```

ATPOL -- Poll for Pending Attention Interruption (O)

ATPOL is used to find out if there is a pending attention (task-asynchronous I/O) interruption (this would be caused by a conversational user hitting the ATTN key); if there is, control is transferred to the address specified by the branch address operand.

Name	Operation	Operand
[symbol]	ATPOL	branch address[,switch]

branch address
specifies a point in a program to receive control if there is an attention pending.

Specified as: An RX address.

switch
specifies the address of a byte whose contents are to be tested. If this byte is set to X'00', the test for a pending attention will be made; if other than X'00', no test will be made for a pending attention and control will pass to the instruction following ATPOL.

Specified as: An RX address.

Default: The test for a pending attention is made. The issuer must have previously defined the symbols ISAAT and ISAATM in his program by copying the DSECT CHAISA (interruption storage area) from the system macro/copy library (SYSMAC).

AUXPG -- Extract Auxiliary Storage Page Counts (O)

AUXPG is used to determine the number of auxiliary drum and disk pages currently available in the system.

Name	Operation	Operand
[symbol]	AUXPG	

Note: There are no operands.

Execution: This macro instruction generates an SVC 230 instruction, which extracts the number of auxiliary drum and disk pages from the auxiliary storage allocation table header. These counts are returned to the user.

Return Data:

Register 0 = Count of unused drum pages.

Register 1 = Count of unused disk pages.

Example: A system programmer, coding a module that services requests for storage, wants to find the number of unused pages that are available for assignment; he can code:

```
STORG     AUXPG
```

AUXSET -- Create Overload/Overdraw Interruption Control Blocks (O)

The AUXSET macro instruction creates interruption control blocks to handle interruptions issued by the resident supervisor when either the task overdraws its auxiliary storage allocation or a system overload condition does not permit allocation of the amount requested.

Name	Operation	Operand
[symbol]	AUXSET	

Note: There are no operands.

Execution: Two interruption control blocks are created in the issuing program's PSECT. These will service interruptions issued by the resident supervisor either (1) to warn a task that it has overdrawn its auxiliary storage allocation, or (2) to cut off a task in an overloaded system in order to relieve a shortage of auxiliary storage. The interruption control blocks are created in the macro expansion and enabled at task LOGON. Servicing of interruption (1) consists of issuing a warning message to SYSOUT. Servicing of interruption (2) consists of issuing a logoff message and a completion code 3 ABEND.

Programming Notes: Use of the AUXSET macro instruction is restricted to the LOGON system module.

Example:

```
NAME      AUXSET
```

AVAUX -- Available Auxiliary Remaining Count (R)

The AVAUX macro instruction is used to compare the amount of auxiliary storage required by a user with the amount available in the system, and to cause a branch to a desired location if enough space is not available. If space is available, the current amount of auxiliary space available is updated.

Name	Operation	Operands
[symbol]	AVAUX	[amount address,]location

amount address
specifies an address containing the number of pages (expressed in binary) of auxiliary storage required by the user.

Specified as: An RX address, or register notation. If register notation is used, the amount address must be loaded into the specified register before issuing the macro instruction.

Default: The value contained in SCMMAV in system common is used; this value represents the maximum amount of auxiliary storage allowed at your installation.

location
specifies the address to which control is passed if sufficient auxiliary storage is not available for the task.

Specified as: An RX address.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding AVAUX. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding AVAUX must be issued with the PRIVILEGED option.

The CHASYS and CHASCM DSECTs must be copied into the issuer's module and covered with base registers.

Execution: If the amount operand is not specified, an installation value is obtained from field SCMMAV in system common. If the amount operand is specified, the value at the specified location is compared with the amount of auxiliary storage currently available, as recorded in the system table CHBSYS. If the amount required is greater than the amount available, control is passed to the indicated location. If the required space is available, the auxiliary space is allocated and the amount currently available updated to reflect this request.

Example: The amount of auxiliary storage required by a task has been prestored in register 5. The macro instruction:

```
CHECK1    AVAUX (5),MSG
```

will update the amount available, or transfer to MSG.

BMSG -- Send BULKIO Message

The BMSG macro instruction is used only by the BULKIO task to send messages from the BULKIO message file to the operator.

Name	Operation	Operand
[symbol]	BMSG	address of message id,[V1,V2...Vn],[message type]

address of message id
specifies the address of a location which contains the external address of a message defined within the BULKIO message handler (CZAWM) module.

Specified as: an absolute address.

Programming Note: The eight byte message is declared as an entry point within the module CZAWM and has the following format: CZAW$## where $ is the BULKIO section id (obtained from use of the ID macro) and ## is a two-digit number from 00-99.

V1,V2...Vn
specifies information that is to be used to complete or alter the message being sent to the operator. The information is substituted for variables (%...%) in the message text.

Specified as: register notation or an absolute address.

message type
specifies where to send the message.

Specified as: register notation, or an absolute expression as follows:

O - write to operator (also write to log)
L - write to log
B - write to operator and log
A - write to operator and log and rje
R - write to log and rje

Default: O

Example:

```
BMSG MSGID,((R2),(R3)),A

MSGID DC V(CZAWA01)
```

Use of this macro would send the message CZAWA01 as defined in CZAWM to the operator, system log, and RJE acknowledgement dataset. Before sending the message the variables pointed to by registers 2 and 3 will be substituted for the %s within the message.

BPKD -- Create a Builtin Procedure Key (O)

Note: This macro has been replaced by BPKDS and must not be used for new code. This documentation is retained only to aid in maintenance of existing programs.

The BPKD macro instruction is used in conjunction with the BUILTIN command to identify a user-coded routine that is to process a user-defined command issued at a terminal. The user-created command must be assigned a name by the BUILTIN command (see Command System User's Guide).

Name	Operation	Operand
symbol	BPKD	entry point name[,(keyword address,...)]

Note: A symbol is required in the name field.

entry point name
specifies the symbolic name of the entry point to the user-coded routine to be executed when a command, named by a BUILTIN command, is issued at the terminal.

Specified as: A symbol (one to eight alphameric characters, the first of which must be alphabetic).

keyword address
specifies the address of a keyword that identifies an operand that is to be specified with the user-created command. Each such keyword defines both the position and the external keyword of a particular operand of the command.

Each user-coded keyword consists of a DC containing the keyword word of the operand being defined. Each such DC must be preceded by the length (one byte) of the character string in the DC (see Example below). Although each DC contains this keyword specification, the actual operand, when specified during command issuance at the terminal, need not be indicated by keyword; it can be specified positionally. The position of each keyword specified within the BPKD macro instruction determines the position of the operand when positional notation is used.

Specified as: A relocatable expression.

Programming Notes: During assembly, the BPKD macro instruction generates a table containing linkage information and parameter storage areas for use by the BUILTIN command associated with the macro instruction. The table contains pointers to the object module that is to process the command. This module is referred to as the command processing routine. The table also contains adcons pointing to each of the keyword defining constants coded by the user. The number of parameters that may be specified in a command is determined by the number of dummy parameters defined in the BPKD macro instruction. Space is reserved in the table so that pointers to the parameter values specified by the user command at the terminal can be recorded.

An ENTRY statement is generated in the table for the symbolic name of the BPKD macro instruction. The recorded entry address is then used to establish linkage between the user-coded module and the BUILTIN command that defines that module as a command processing module and associates it with a particular command name. When the named command is issued, pointers to the actual values of any positional or keyword parameters specified at the terminal are placed in the reserved storage areas within the generated table in the PSECT that contains the BPKD macro instruction; register 1 is then set to point to these pointers so that the data can be referred to by the command processing module. If any command operand defined by BPKD is omitted when the command is issued at the terminal, these pointers are set to point to any default values; if none exist, they are set to zeros. Control is then passed to the command processing module which is executed using the necessary parameter values to accomplish the desired goal.

If no parameter names have been defined in the BPKD macro instruction (indicating that no parameters are to be specified when the command is issued), the expansion of the macro instruction does not set up dummy parameter areas or allow specification of parameters within the command. Examples of the table generated by BPKD are shown below.

The BPKD macro instruction must be supplied in a PSECT and must have any expected parameters defined therein. If the command processor is assembled in a module different from the one containing the BPKD macro instruction, its entry point and PSECT symbols must be used as arguments of an EXTRN statement in the assembly containing the BPKD macro instruction.

When a user wants to provide parameters for the command he is creating, he must provide code in the PSECT (for each such parameter) indicating a keyword to be associated with that parameter and the length of

that character string. The parameter addresses of the BPKD macro instruction must also be the name fields of the user-defined character strings (that is, PAR1 DC C'KEYWORD', where PAR1 is a parameter address for BPKD). It is these parameter strings or keywords that will be associated with the parameters of a calling command when the command is issued.

Example: If a user wants to execute the object program instructions in a particular CSECT when a command called TROT is issued at the terminal, he could indicate this by specifying the following in a PSECT associated with the command-processing module:

```
BPKLABEL  BPKD   EPPROC,(DPAR1,DPAR2)
            .
            .
            .
          DC     AL1(L'DPAR1)          CODED BY USER FOR ALL
DPAR1     DC     C'KEYWORD1'             DESIRED COMMAND
          DC     AL1(L'DPAR2)            OPERANDS
DPAR2     DC     C'KEYWORD2'
```

At the terminal:

```
            .
            .
BUILTIN   TROT,BPKLABEL
TROT      KEYWORD1=70,KEYWORD2=5000
            .
            .       (or, if positional parameters
            .       are desired)
            .
TROT      70,5000
            .
```

When TROT is issued, pointers to keyword and positional parameters are placed in areas reserved within the BPKD macro expansion, making them available to the command expansion routine located at entry point CSECTMOD. The routine at CSECTMOD is entered, with register 1 pointing to a list, which contains fullword address pointers to the actual parameter values entered at the terminal (the length of these actual values can be found in the byte preceding the value location); then the routine is executed and control is returned to the user terminal.

BTRUBL -- Set Last Called ID into BULKCOMM and S-entry Table

The BTRUBL macro instruction moves an 8 byte external call id into the BULKCOMM and S-entry tables. This external call id will be used by the BULKIO abend recovery routine (CZAWA) for purposes of recovery when an error has occurred and ABEND has been called.

Name	Operation	Operand
[symbol]	BTRUBL	continuation address, module last called

continuation address
specifies the point at which processing will continue when control is returned.

Specified as: an RX address

module last called
specifies the name of the external routine called by a BULKIO module before ABEND was called.

Specified as: an 8 character absolute expression of which the first two characters must be either "V-" (for a VAM call) or "M-" (for an MSAM call).

Example:

```
     BTRUBL  EODAD,=CL8"V-GET"
+       MVC   SETYCONT,=A(EODA)      save continue address
+       MVC   SETCALL,=CL8"V-GET"
+       MVC   BLTCALL,=CL8"V-GET"
```

Initialization: You must provide the following DSECTS in order to use this macro instruction: CHABCT & CHASET.

CANCL -- Cancel Realtime Interruption (O)

The CANCL macro instruction permits privileged resident programs to specify that a specific realtime interruption request (for example, one requested by the SETTIMER system macro instruction) be canceled.

Name	Operation	Operand
[symbol]	CANCL	[return parameter],interruption routine adcon

return parameter
specifies the return parameter previously specified by a SETTIMER macro.

Specified as: Register notation (0 through 15). The return parameter must be loaded into the specified register before issuing the macro instruction.

interruption routine adcon
specifies the entry point of a module, previously specified by SETTIMER, which would have received control at time of the interruption if it was not being canceled. If this parameter is set to X'FFFFFFFF', all interrupts for the given return parameter are canceled. This type of call is used by DELETE TSI (CEAMD).

Specified as: A symbol, or register notation (1 through 15). If register notation is used, the entry point must be loaded into the specified register before issuing the macro instruction.

Initialization: All modules using the CANCL macro instruction must first have a copy of CHAPSA and a USING CHAPSA statement.

Execution: CANCL checks whether the interruption routine's adcon was specified; if not, an error is declared. Type-1 linkage is used to branch to the Set Realtime Interruption routine which deletes the realtime value associated with that return parameter from the realtime-interruption-pending queue in the resident supervisor.

Return Data: Register 15 is set to X'0C' for normal returns.
Register 15 is set to X'08' if there was no entry to CANCL.

Example: The realtime interruption previously scheduled by a SETTIMER macro instruction for the TSI whose address is in TSIREG, with the interruption servicing routine of CEAMA, is canceled by issuing:

```
OKAYA       .
            .
            CANCL   TSIREG,CHANA
            .
```

CHANGE -- Change Schedule Table Entry (R)

The CHANGE macro instruction permits you to alter the schedule table entry (STE) level of a task to desired level and to retrieve the current level. If the new level is valid, it is placed in the task's TSI.

Name	Operation	Operand
[symbol]	CHANGE	level

level
: specifies the address of a fullword containing the new schedule level.

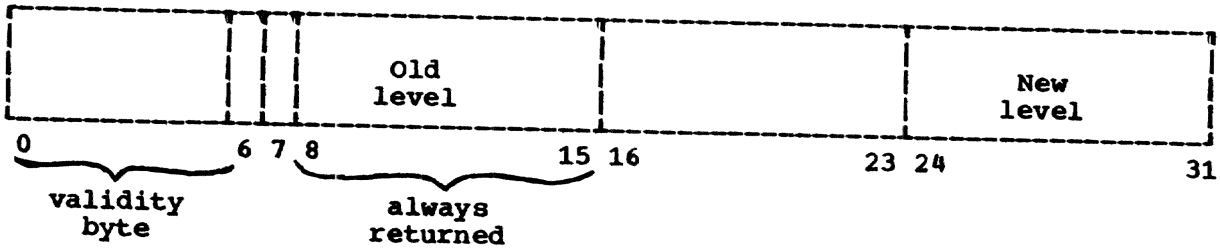
  Specified as: An RX address, or register notation. Execution time is saved if register 15 is specified.

  Note: If register notation is used, the new schedule level value itself (not its address) must be loaded into the specified register before issuing the macro instruction.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding CHANGE. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding CHANGE must be issued with the PRIVILEGED option.

Execution: The new schedule level is passed to the resident supervisor in register 15. This level represents an index position (0-255) within the resident supervisor's schedule table. When the supervisor receives the interruption, a check is made to determine if the new schedule leve- lis valid (that is, within the schedule table limits and a nonzero entry). If valid, the task's schedule level is changed to the value specified in register 15. If not valid, an error indicator is set in register 15. The current schedule level is always returned in register 15.

Return Data: The format of the return information in register 15 is:



Validity byte values:

Setting	Meaning
All zeros	New level is valid.
Bit 6 (on)	Outside the limits of the schedule table.
Bit 7 (on)	Zero entry was specified.

Example: Suppose you wish to change the current level of a task to index position 10. You write:

```
        LA      15,10
CHGEL   CHANGE  (15)
```

CHDINNRA -- Generate Type-1 or Type-2 Linkage (O)

The primary function of the CHDINNRA inner macro instruction is to generate a type-1 or a type-2 linkage or to generate a linkage by means of an SVC. It may also be used to load registers 0 and 1 with parameters or load the second element of the MF sublist in the E-form of the S-type macro instruction.

Name	Operation	Operand
[symbol]	CHDINNRA	[parameter 1],[parameter 2], [([sublist element 1],[sublist element 2])] [,enter code,macro code]

parameter 1
specifies a parameter to be loaded into register 1.

Specified as: An RX address.

Default: It is assumed that any desired parameter is already loaded into register 1.

parameter 2
specifies a parameter to be loaded into register 0.

Specified as: An RX address.

Default: It is assumed that any desired parameter is already loaded into register 0.

sublist element 1
is the first element of a two-element sublist. If specified by itself, it designates the entry point for a type-1 linkage. If specified together with the second element, it indicates the relative byte location within the DCB at which OPEN has placed the R-value.

Specified as: The name of an entry point (if sublist element 2 is omitted), or a number or absolute expression (if sublist element 2 is included).

Default: Sublist element 2 is interpreted as specifying the operand of an SVC instruction. If sublist element 2 is also omitted, no linkage is generated.

sublist element 2
is the second element in the sublist. If it is specified by itself, it is interpreted as the integer specified in the operand field of an SVC instruction. If specified together with sublist element 1, it is interpreted as the relative byte location of the V-value within the DCB.

Specified as: A number or absolute expression.

Default: Sublist element 1, if present, is interpreted as an entry point for type-1 linkage. If sublist element 1 is also omitted, no linkage is generated.

enter code
specifies the enter code (see Appendix A) to be used in generating a type-2 linkage.

Specified as: A number or an absolute expression.

macro code
specifies a code to be stored in the macro code field (MACRF) of the DCB.

Specified as: One of the codes explained under the MACRF= operand of the DCB macro instruction (see Assembler User Macro Instructions).

CAUTION: Since the omission of both sublist elements results in no linkage being generated, you must furnish at least one sublist element when using CHDINNRA to generate a type-2 linkage. In addition to providing the enter code operand, you must also provide a dummy entry point in a sublist element.

The macro code must be specified only when the outer macro instruction has a DCB address to be placed in register 1. Also, parameter 1 and parameter 2 may be used only when the value to be loaded into the appropriate register can be used as the second operand of an LA instruction.

Example A: The macro instruction:

```
.LINK CHDINNRA ,,(CZCXYZ),X'FF'
```

results in the generation of either a type-1 linkage to CZCXYZ or a type-2 linkage to that routine with an enter code of 255, depending on the privilege class of the issuing module. This determination is made by testing the value in &CHDCLS.

Example B: The coding:

```
EFORM CHDINNRA MF(2)
```

results in the second element of the MF=operand being put in register 1.

Example C: The macro instruction:

```
ERROR CHDINNRA ,,(,254)
```

results in the generation of SVC 254.

CHGVLOCK -- Exchange VM Locks (O)

The CHGVLOCK macro instruction is used to exchange VM Locks when processing a chain of individually-locked control blocks.

Name	Operation	Operand
[symbol]	CHGVLOCK	log1,log2

log1,log2
identify two VM Locks to be exchanged.

Specified as: the symbols naming LOGVLOCK macros.

Execution: The active areas of the specified VM Lock Anchors will be exchanged.

CAUTION: This macro must be protected from task interrupts by ITI/PTI.

Programming Note: Refer to VM Locking in Section 3.

CLOSE (MSAM) -- Disconnect Data Set From User's Problem Program (S)

The CLOSE macro instruction disconnects one or more data sets from the user's problem program.

Standard form:

Name	Operation	Operand
[symbol]	CLOSE	(dcb address,...)

L-form:

Name	Operation	Operand
symbol	CLOSE	[(dcb address,...),]MF=L

Note: A symbol is required in the name field of the L-form.

E-form:

Name	Operation	Operand
[symbol]	CLOSE	[(dcb address,...),]MF=(E,list)

Note: If a dcb address is not specified in the L-form, it must be provided with the E-form. If specified in both, the E-form operand overlays the corresponding operand in the L-form. No more dcb addresses may be specified in the E-form than were specified in the L-form.

dcb address
specifies the address of the data control block opened for the data set whose processing is to terminate.

Specified as: In the standard and L-form, as a relocatable expression; in the standard and E-form, in register notation (2 through 12); in the E-form only, also as an RX address. If register notation is used, the dcb address must be loaded into the specified register before issuing the macro instruction.

CAUTION: The following errors cause the results indicated:

Error	Result
Closing data control block that is already closed.	No action
Closing when dcb address operand does not specify address of data control block.	Task terminated
Closing data control block containing invalid DSORG specification.	Task terminated

Programming Notes: You may specify any number of data control block addresses in the CLOSE macro instruction. This facility makes it possible to close data control blocks and their associated data sets in parallel.

In most instances, the FINISH macro instruction should precede CLOSE (see the explanation of the FINISH macro instruction in "Macro Instructions for MSAM"), since you cannot be informed from CLOSE of errors that may have occurred in processing the last output buffer page. Additionally, the use of CLOSE without a preceding FINISH that returned a normal completion code would cause the task to wait until the I/O operation for that DCB is complete.

The CLOSE macro instruction for MSAM releases all the storage area that was used for MSAM. The options of REREAD and LEAVE are ignored. If the FINISH macro instruction does not precede the CLOSE and if the DCB INHMSG=0, a message is written to the operator to remove the data set from the device. If the DCB COMBINE flag is set and if a FINISH does not precede the CLOSE, a card is read from the reader on the same 2540 as the selected punch and stacked in pocket 3.

CLRVLOCK -- Clear a VM Lock (O)

The CLRVLOCK macro instruction is used to open a VM Lock during abnormal termination recovery by a module which uses other xxxVLOCK macros.

Name	Operation	Operand
[symbol]	CLRVLOCK	log

log
specifies the VM Lock to be cleared.

Specified as: the symbol naming a LOGVLOCK macro.

Execution: The specified VM Lock Anchor is examined, and the equivalent of OPNVLOCK is executed if the lock is indicated as "set".

CAUTION: This macro must be protected from task interrupts by ITI/PTI unless used in a standard ABEND Interlock Release routine, in which case the AIR call will be made with task interrupts inhibited and no ITI/PTI should be attempted.

Programming Note: Refer to VM Locking in Section 3.

CNSEG -- Connect Segment to Shared Page Table (R)

The CNSEG macro instruction is used by the system to connect a segment table entry to a shared page table.

Name	Operation	Operand
[symbol]	CNSEG	[segment number,shared page table number]

segment number
specifies the segment table entry to be connected to the shared page table.

Specified as: An absolute expression or register notation (1 through 12). If this operand is specified, the shared page table number operand must also be specified. If register notation is used, the segment number should occupy the high-order halfword of the register.

Default: It is assumed that the issuer has placed the segment number in the high-order halfword of register 1.

shared page table number
specifies the number of the shared page table to which the segment is to be connected.

Specified as: An absolute expression or register notation. This operand must be specified if the segment number operand was specified. If register notation is used, only one register may be specified for both this operand and the segment number, and the shared page table number must be placed in the low-order halfword of the register.

Default: It is assumed that the issuer has placed the shared page table number in the low-order halfword of register 1.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding CNSEG. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding CNSEG must be issued with the PRIVILEGED option.

Execution: The shared page table number in the low-order halfword of register 1 is used to search the task's auxiliary segment table. If an auxiliary segment table entry is already connected to the shared page table, its segment number replaces the high-order halfword of register 1.

If no auxiliary segment table entry is connected to the specified shared page table, the segment table entry indicated by the high-order halfword of register 1 is set not available; its auxiliary segment table entry is marked assigned and shared, and the shared page table number in register 1 is inserted into the auxiliary segment table entry.

Programming Note: The CNSEG macro instruction is used by the privileged system to keep track of a task's shared page and segment tables. For example, a CONNECT request for virtual storage allocation (VMA) causes a CNSEG macro instruction to be issued. The system programmer should use the CONNECT entry point (CZCGA7) in VMA to accomplish this function.

Example: Suppose shared page table number 3 is to be connected to segment 12. The macro instruction might be written:

```
RJG   CNSEG    12,3
```

This would generate (showing you how the number in register 1 is obtained):

```
RJG   DS       0H
&A3   SETA     3+12*65536
```

```
L       1,=F'&A3'
SVC     238
```

Shared page table 3 is connected to segment 12 and the high-order halfword of register 1 is left unchanged (as 12) to indicate the actual segment to which the shared page table was connected.

CRTSI -- Create Task Status Index (R)

The CRTSI macro instruction allocates storage for a TSI and initializes it for a new task if the system limit on TSIs has not been reached.

Name	Operation	Operand
[symbol]	CRTSI	

Note: There are no operands.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding CRTSI. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding CRTSI must be issued with the PRIVILEGED option.

Execution: A new task status index is created if the system TSI limit has not been reached.

Return Data: The task identification of the new task is returned in register 0; if the system TSI limit has been reached, register 0 is set to 0.

Example: If you want to create a TSI, you might write:

```
XYZ     CRTSI
```

This would generate:

```
XYZ     SVC 253
```

Note: This SVC must be used in conjunction with VSEND.

CSEG -- Connect Named Segment (0)

The CSEG macro instruction transfers control to the connect named segment program in the resident supervisor. An attempt will then be made to connect the specified disconnected segment group.

Name	Operation	Operand
	CSEG	

Note: There are no operands.

Initialization: Before executing CSEG, the issuing program should have set up the following parameter area:

```
CHANSG   DSECT ,          COMMON NAMESEG PARAMETER LIST
         DS    0F
NSGSVC   DS    H          SVC
         DS    XL2        RESERVED

NSGRNA   DS    XL8        RESERVED SEGMENT GROUP NAME
NSGDNA   DS    XL8        DISCONNECTED SEGMENT GROUP NAME
```

```
NSGVMA   DS    A         VIRTUAL STORAGE ADDRESS OF SEG GROUP

NSGLNG   DS    H         LENGTH OF NAMED GROUP

NSGFLI   DS    XL1       INPUT FLAGS
NSGFLO   DS    XL1       OUTPUT FLAGS
NSGDNGM  EQU   X'80'     DNAME SPECIFIED
NSGRNGM  EQU   X'40'     RNAME SPECIFIED
NSGADGM  EQU   X'20'     ADDRESS SPECIFIED
NSGBNDM  EQU   X'10'     MODE=BOUND
NSGLNGM  EQU   X'08'     LENGTH SPECIFIED

         DS    F         RESERVED
NSGLTH   EQU   *-CHANSG  LENGTH OF PARAMETER LIST
```

CSEG must be the object of an execute instruction and be fullword aligned.

Execution: This macro instruction passes control to the resident supervisor module CEAP5 via SVC 183. An attempt will then be made to reconnect the specified disconnected segment group.

Programming Note: The system programmer should use the CONSEG macro instruction described in Assembler User Macro Instruction, GC28-2004.

CVT -- Activate Communications Vector Table

The CVT macro sets a GLOBAL SETB symbol CHDCVT used by system macros to determine if the user has furnished a base register for the system CVT control block. The CVT macro also does a copy of the CVT dsect CHACVT if the user has not provided one. If the user has not provided a base register for the CVT (CVT NO), then the system macros use Register 15.

Name	Operation	Operand
[symbol]	CVT	{YES|NO}

YES|NO
indicates whether or not the user has provided a 'USING CHACVT,Rx' statement assigning a base register for the CVT control block.

Default: NO.

CAUTION: CVT sets the GLOBAL SETB symbol CHDCVT. If CVT YES, CHDCVT is set to a 1.

Programming Notes: The PSA dsect CHAPSA must be copied by any assembly using any supervisor macros. Otherwise, the macros will generate error statements when assembled.

The pointer for the CVT control block is in the PSA and is symbolically addressed as PSACVT.

Example:

```
        USING CHAPSA, R0      assign base register to PSA
        L R12,PSACVT          load address of CVT
        USING CHACVT,R12      assign base register to CVT
        CVT YES               tells system base register is assigned
```

DCB (MSAM) -- Set Up Data Control Block (O)

An explanation and the general format of the DCB macro instruction is contained in Assembler User Macro Instructions. The operands available only to the TSS user with privilege class E (system monitor) for use with MSAM are described below and supplement the operands described in Assembler User Macro Instructions.

You can refer to the information stored in the data control block by means of the DCBD macro instruction, which is also described in Assembler User Macro Instructions.

Information regarding a data set may be provided through the DDEF command and macro instruction and by your program itself (by placing information in the data area created by your DCB macro instruction) in addition to the information furnished at the time the DCB macro instruction is issued. Figure 23 illustrates possible sources of information for various fields.

	Alternate Sources			
DCB Field	Your Program Prior to OPEN	DDEF Command and Macro Instruction	DCB Macro Instruction	Your Program After OPEN
DSORG	X			
MACRF	X	X	X	
DDNAME	X		X	
DEVD	X	X	X	
PRTSP[1]	X	X	X	X[2]
MODE[3]	X	X	X	X[2]
STACK[3]	X	X	X	X[2]
RECFM	X	X	X	X[2]
LRECL	X	X	X	X[4]
POCKET	X		X	X[2]
RETRY	X		X	X[2]
SUR				X[5]
INHMSG	X	X	X	X
FIP				X[6]
COMBINE	X		X	
FORMTYPE	X		X	X[2]

[1]Checked only if DEVD specifies a printer (PR).

[2]Checked only if a FINISH macro instruction has been executed and a return code other than 4 was provided, and if no GET or PUT macro instruction has been executed after the FINISH.

[3]Checked only if DEVD specifies a card punch (PC) or a card reader (RD).

[4]For format-F records, footnote 2 applies.

[5]Only if a SETUR macro instruction has been executed and a return code of 4 was provided.

[6]Only if a FINISH macro instruction has been executed and a return code of 4 was provided.

Figure 23. Sources of DCB information for MSAM

The data control block (DCB) for MSAM includes five fields (RETRY, COMBINE, POCKET, FORMTYP, and INHMSG) that are not required in the DCBs for the other access methods. These fields, as well as others pertinent

to the access method, may be filled using the DCB macro instruction, as follows:

Name	Operation	Operand
symbol	DCB	DSORG=MS[,MACRF={G|P}][,DDNAME=name] [,DEVD=code][,RECFM=code][,LRECL=record length] [,RETRY={N|U}][,COMBINE={Y|N}][,POCKET={1|2|ORG}] [,FORMTYP={F|S|D}][,INHMSG={Y|N}]

Note: If the MACRF, DDNAME, DEVD, RECFM, LRECL, RETRY, POCKET, and FORMTYP operands are not specified, the information must be furnished by another source described in Figure 23.

DSORG=
specifies the organization of the data set.

Specified as: For MSAM, the only acceptable code is MS.

MACRF=
specifies the type of macro instructions to be used in processing the data set.

Specified as: For MSAM, only the options G and P (for the GET and PUT macro instructions) are permitted.

DDNAME=
specifies the symbolic data definition name associated with the data set.

Specified as: A name of three to eight alphameric characters. This name must be the same as the ddname of the DDEF command or macro instruction associated with this data control block. Each data set to be processed requires one DCB macro instruction and one DDEF command or macro instruction.

DEVD=
specifies the type of device on which the data set resides.

Specified as: The three options available for MSAM are PC, PR, and RD (for the card punch, printer, and card reader). Additional keyword operands are available, as shown below, to provide device-dependent information to the device-dependent parameter bytes in the data control block.

Code	Meaning	Additional Keyword Operands
PC	Card punch	[,MODE={C|E}] [,STACK={1|2|3}]
PR	Printer	[,PRTSP={0|1|2|3}]
RD	Card reader	[,MODE={C|E}] [,STACK={1|2}]

MODE=
specifies the mode of operation for a card punch or card reader.

Specified as: For MSAM, the available options are: C (column binary mode) or E (EBCDIC mode). The value of the MODE field may not be altered after the DCB is opened except between a completed FINISH and the next GET or PUT macro instruction.

Default: E

STACK=
specifies which stacker bin is to receive the record.

Specified as: 1, 2, or 3. Stacker bin 3 may be specified only if the punch (PC) is specified. The STACK field is ignored if A or M is specified in RECFM at OPEN time.

Default: 1

PRTSP=
specifies the number of line spaces after printing. The field will be ignored if A or M is specified in the DCB RECFM field.

Specified as: 0, 1, 2, or 3.

Default: 1

RECFM=
specifies the characteristics of the records in the data set.

Specified as: Any of the following order are valid: {F|V}[B][A|M]. B, however, will be ignored. Any other record format designations will abnormally terminate the task.

Control Characters A|M: As an optional feature, records may include a control character in each logical record. This control character will be recognized and processed if a data set is being written to a printer or punch by MSAM. This character is provided by the user as the first byte of the logical record. Two options are available:

- FORTRAN* Code (A) - The user may choose to specify this code rather than the machine code. The control byte must appear in each logical record if this option is chosen.

- Machine Code (M) - The user may specify in the data control block that the machine code control character has been placed in each logical record. The byte supplied by the user must contain the bit configuration specifying a write and the desired carriage or stacker select operation. (This permits independent carriage and stacker select operations.)

LRECL=
specifies the record length or, for format-V, the maximum record length.

Specified as: For format-F records, the length in bytes. This length must include the control character for an output data set if (A|M) is specified in RECFM. The length may not exceed 80 bytes for reading in EBCDIC mode and 160 bytes for column binary mode. For an output data set on a printer, the maximum is 133 bytes; and on a card punch, 81 bytes for EBCDIC and 161 bytes for column binary (the additional byte for output data sets is for the control byte only if A or M is specified in the RECFM).

For format-V records, the maximum length in bytes of a logical record. LRECL may be modified after the DCB is opened at any time. The length can not exceed 84 bytes for reading in EBCDIC mode and

---

*Formerly called ASA or extended USASI code, this code contains control characters defined by American National Standard FORTRAN, ANSI X3.9-1966, hereinafter referred to as FORTRAN control characters.

164 bytes for column binary mode for an input data set. For an output data set on a printer, the maximum is 137 bytes; and on a card punch, 85 bytes for EBCDIC and 165 bytes for column binary. The additional byte for output data sets is for the control byte only if A or M is specified in the RECFM. The four or five control bytes of a format-V logical record are not punched or printed. A format-V logical record will have five control bytes when a FORTRAN control character is specified.

RETRY=
specifies the error retry for the 2540 (reader only).

Specified as: The available options are N or U, where N indicates no retry and U indicates unlimited retry.

Default: U

COMBINE=
specifies that a card is to be read in when FINISH is issued during a PUNCH job, effectively separating different punch jobs. Each time a FINISH or CLOSE macro instruction is issued, a card is read from the reader and stacked in pocket 3.

Specified as: Y or N

Default: N

Note: The STACK option chosen for the punch must be 3 if COMBINE is Y; otherwise, the task will be abnormally terminated.

POCKET=
specifies error card stacker bin (for the card reader on the 2540 only).

Specified as: The available options are ORG, 1, and 2, where ORG means stack as if no error occurred, 1 means stacker bin 1, and 2 means stacker bin 2.

Default: 1

FORMTYP=
specifies how printer error retry is to be performed.

Specified as: D, F, and S. The meaning of these codes is described below.

D (storage dump type): Attempt write retry (no skip or space) three times before the DECB is set complete with error. If one of three retries is successful, use the strike-out character (X) to strike out all print positions and reprint the line. If three lines on one page have been struck out, eject to channel 1 and continue.

F (form-sensitive type): Attempt write retry (no skip or space) three times before the DECB is set complete with error. If one of the retries is successful, eject to channel 1, tell the operator to mark the error page, wait for acknowledgement, and then rewrite the entire page. The page is rewritten starting from the previous skip to channel 1. Note: FORTRAN (ASA) or machine control characters must be used with F-type forms.

S (sequence-sensitive type): Attempt write retry (no skip or space) three times before the DECB is set complete with error. If one of the retries is successful, tell the operator to mark error page, wait for acknowledgement, and then continue with the

same output page. Note: FORTRAN (ASA) or machine control characters must be used with S-type forms.

Default: D

INHMSG=
specifies whether or not messages to the operator to remove the data group when a CLOSE or FINISH macro instruction is issued are to be suppressed.

Specified as: Y (yes) or N (no)

If the N option is chosen and CLOSE or FINISH is issued, a message is sent to the operator to remove the data group from the device. When the operation indicated by the WTO is complete, the operator is some instances will be requested to generate an asynchronous interruption by changing the status of the device from not ready to ready. This bit, DCBINHMS, may be dynamically modified by use of the DCBD macro instruction.

Default: N

DCLASS -- Specify Privilege Class (O)

The DCLASS macro instruction declares the user's intent to assemble privileged or nonprivileged code by setting the global SETB symbol, &CHDCLS, to 1 (privileged) or 0 (nonprivileged). Unlike other macro instructions, DCLASS is used only during assembly. It directs macro definitions that check the symbol &CHDCLS to generate privileged or nonprivileged code.

Name	Operation	Operand
[symbol]	DCLASS	[privilege class]

privilege class
specifies whether the code following this macro instruction is to be assembled as privileged or nonprivileged.

Specified as: PRIVILEGED (privileged code) or USER (nonprivileged code).

Default: USER. If no DCLASS is issued it is assumed the code is nonprivileged.

CAUTION: Before issuing a macro instruction whose corresponding macro definition refers to the &CHDCLS global symbol, a user must be sure the symbol is set properly. The required settings for the various TSS macro instructions that use this mechanism are indicated with their descriptions. DCLASS settings required to issue macro-generated SVC instructions are shown in Appendix B.

DELET -- Enter Delete Program (O)

The DELET macro instruction causes control to pass to the dynamic loader's Delete routine via the task monitor.

Name	Operation	Operand
[symbol]	DELET	

Note: There are no operands.

Execution: A task-SVC interruption is created to transfer control to the task monitor. Control is then transferred to the dynamic loader's Delete routine. (See Assembler User Macro Instructions for a description of the DELETE macro instruction.)

Example: If you want your program to enter the Delete program within the dynamic loader, you would not use the ENTER mechanism; you could write:

```
          EX      0,NAME
          B       AWAY
NAME      DELET
          DC      CL8'DESTRYME'
          DC      X'0000'
```

DELET would expand as:

```
NAME      SVC     123
```

DELPG -- Delete Virtual Storage Pages (R)

The DELPG macro instruction is used by the system to delete contiguous pages from a task's virtual storage and, when possible, to delete the associated page table entries.

Name	Operation	Operand
[symbol]	DELPG	[page count] [,first page address]

page count
specifies the number of contiguous virtual atorage pages to be deleted.

Specified as: an absolute expression or register notation.

Default: If the operand is omitted, it is assumed that the page count has been placed in register 0.

first page address
specifies the address of the first virtual storage page to be deleted.

Specified as: an RX address or register notation.

Default: If the operand is omitted, it is assumed that the first page address has been placed in register 1.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding DELPG. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding DELPG must be issued with the PRIVILEGED option. Before the DELPG macro instruction is executed, the issuing program must set up the first byte of register 15 with one of the following codes:

Code	Meaning
00	user requested delete
02	system requested delete

Execution: The contiguous pages, beginning at the address contained in register 1 and equal to the count in register 0, are deleted from the issuing task's virtual storage. Main storage and paging storage space in use for the released pages are freed for reallocation. If an entire segment is deleted, the auxiliary segment table entry is marked unassigned, the segment table entry is marked not available, and an indicator is set to represent the deleted segment. The space occupied by the page table entries and external page table entries is returned for reallocation. If the auxiliary segment table entry is marked shared, the entry corresponding to the segment in the resident shared page index is deleted.

Programming Notes: The DELPG macro instruction is issued by the privileged system to keep track of a task's page, segment, and shared page tables. For example, a FREEMAIN request for virtual memory allocation (VMA) causes a DELPG macro instruction to be issued. The system programmer should use the FREEMAIN macro instruction to accomplish this function.

The DELPG macro instruction can be used for deleting both unshared and shared pages.

Example: Suppose three pages of virtual storage originally allocated by a system program, starting at ABCXYZ, are to be deleted. The macro instruction might be written:

```
.
.
.
LA     0,3             PAGE COUNT
LA     1,ABCXYZ        FIRST PAGE ADDRESS
LA    15,X'02'         SYSTEM REQUESTED DELETE
SLL   15,24            *
DELPG
.
.
.
```

DEQGQE -- Dequeue GQE from SCAN Table

DEQGQE sets up the parameter list and calls CEAJDE to dequeue a GQE from the SCAN table.

Name	Operation	Operand
[symbol]	DEQGQE	GQE

GQE
address of the GQE to be dequeued.

Specified as: the name of a symbol defined as a fullword, or Register 2 through 12 or 1.

CAUTIONS: The PSA dsect CHAPSA must have been copied and assigned a base prior to the use of this macro. The expansion of this macro is affected by use of the CVT macro prior to the use of this macro.

Examples:

(1) Register notation; CVT YES is assumed

```
DEQGQE (1)
* L  15,CVTJDE
* BALR 14,15
```

(2) Symbolic name; CVT YES is assumed

```
 DEQGQE TCWGQE        where TCWGQE is defined as a fullword
   .
   .
   .
+ CHDVAL TCWGQE,1
+  L 1,TCWGQE
+  L 15, CVTJDE
+  BALR 14,15
```

(3) Symbolic name; CVT NO is assumed

```
 DEQGQE TCWGQE        where TCWGQE is defined as a fullword
   .
   .
   .
+  CHDVAL TCWGQE,1
+  L 1,TCWGQE
+  L 15,PSACVT
+  L 15,CVTJDE-CHACVT(,15)
+  BALR 14,15
```

DISABLE -- Disable System Interrupts

DISABLE permits the user to "disable" system interrupts and address translation. It sets a new system mask, and if desired by the user, saves the previous system mask in a specified area.

Name	Operation	Operand
[symbol]	DISABLE	[AREA=] [,IO=] [,EXT=] [,PER=] [,TRAN=]

AREA
area in which the current system mask is to be stored

Specified as: an RX address or register notation

Default: Current system mask is stored in a field in the PSA-PSASM

IO=
I/O interrupts

Specified as:
Y - disable I/O interrupts.
N - status of I/O interrupts is unchanged.

Default: Y.

EXT=
external interrupts.

Specified as:
Y - disable external interrupts.
N - status of external interrupts is unchanged.

Default: Y.

PER=
Program Event Recording Interrupts.

Specified as:
Y - disable program event recording interrupts.
N - status of program event recording interrupts is unchanged.

Default: Y.

TRAN=
address translation.

Specified as:
Y - disable address translation.
N - status of address translation is unchanged.

Default: Y.

Execution: DISABLE uses the S/370 instruction STNSM.

Return Data: AREA, if defined, contains the previous system mask after execution of the DISABLE.

Examples: To disable only I/O interrupts, you may write:

```
DISABLE IO=Y,EXT=N,PER=N,TRAN=N
```

To disable all interrupts and later restore the system mask to its previous configuration, you may write:

```
         DISABLE AREA=AREA      disable all interrupts and
                                save current system mask
         SSM SAVE               restore old system mask

SAVE DC X'00'                   area to save system mask
```

DISCSEG -- Disconnect Segment Group (0)

This macro instruction is completely documented in the Assembler User Macro Instruction manual, except for one operand that is available only to a systems programmer. The definition and specification for only that one operand are given below, but for continuity, the metalanguage format that follows shows all the operands.

L-form

Name	Operation	Operand
Symbol	DISCSEG	[DNAME=,LENGTH=,BOUND=,RNAME=,RELEAS=,] MF=L

E-form

Name	Operation	Operand
[symbol]	DISCSEG	[DNAME=,LENGTH=,BOUND=,RNAME=,ADDRESS=,RELEAS=,] MF=(E,LIST)

Standard-Form

Name	Operation	Operand
[symbol]	DISCSEG	[DNAME=,LENGTH=,BOUND=,RNAME=,ADDRESS=,RELEAS=,]

Note: all operands are keyword.

RELEAS=
specifies whether the reserved segment GETMAIN option is enforced while the named segment is disconnected from the task's address space (Refer to the GETMAIN and RSVSEG macros).

Specified as:

Y - the reserved segment GETMAIN option is not enforced.

N - the reserved segment GETMAIN option remains in force.

Default: N

Note: this option is available to privileged class modules only.

DLINK -- Transfer to Dynamic Loader for External Symbol Resolution (O)

The DLINK macro instruction provides external symbol resolution (explicit loading) and may also provide transfer of control to a program (explicit linking).

Name	Operation	Operand
[symbol]	DLINK	

Note: There are no operands.

Execution: The resident supervisor creates a task-SVC interruption to transfer control to the task monitor. Control is transferred to the dynamic loader's DLINK (dynamic linkage) routine. DLINK can be used for explicit linking (external symbol resolution and transfer of control to loaded program) or explicit loading (no transfer of control). DLINK must be the subject of an execute instruction. (Also see CALL, LOAD, ARM, and ADCON in Assembler User Macro Instructions.)

Example: Suppose you want to dynamically load a program called HELP and have control transferred to its entry point, BEGIN. You might write:

```
LOAD      EX      ADCNGRP
          B       AWAY
ADCNGRP   DS      0F
          DLINK
          DC      X'0100'        OPTIONS:  LOAD AND TRANSFER
          DC      CL8'BEGIN'
          DC      2F'0'
```

This causes the dynamic loader to receive control from the task monitor via the supervisor; it will then load HELP and transfer to BEGIN.

DLTSI -- Delete Task Status Index (O)

The DLTSI macro instruction deletes the specified TSI and removes its associated task from the system.

Name	Operation	Operand
[symbol]	DLTSI	

Note: There are no operands.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding DLTSI. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding DLTSI must be issued with the PRIVILEGED option.

Execution: The task issuing the SVC is eliminated from the system. All nonshared pages of main storage and paging storage used by the task are returned for reallocation. All storage required for table entries pertaining to the task in the resident supervisor is released.

Example: To release all the resources a task is using -- logically eliminating the task -- this might be written:

```
RJG    DLTSI
```

Note: This SVC is the last step in a sequence required for removing a task. Other steps include closing data sets and releasing external storage and devices.

DSEG -- Disconnect Named Segment (O)

The DSEG macro instruction transfers control to the disconnect named segment program in the resident supervisor. An attempt will then be made to name and disconnect the specified segment group.

Name	Operation	Operand
	DSEG	

Note: There are no operands.

Initialization: Before executing DSEG, the issuing program should have set up the following parameter area:

```
CHANSG   DSECT ,          COMMON NAMESEG PARAMETER LIST
         DS    0F
```

```
NSGSVC   DS    H          SVC
         DS    XL2        RESERVED

NSGRNA   DS    XL8        RESERVED SEGMENT GROUP NAME
NSGDNA   DS    XL8        DISCONNECTED SEGMENT GROUP NAME

NSGVMA   DS    A          VIRTUAL STORAGE ADDRESS OF SEG GROUP

NSGLNG   DS    H          LENGTH OF NAMED GROUP

NSGFLI   DS    XL1        INPUT FLAGS
NSGFLO   DS    XL1        OUTPUT FLAGS
NSGDNGM  EQU   X'80'      DNAME SPECIFIED
NSGRNGM  EQU   X'40'      RNAME SPECIFIED
NSGADGM  EQU   X'20'      ADDRESS SPECIFIED
NSGBNDM  EQU   X'10'      MODE=BOUND
NSGLNGM  EQU   X'08'      LENGTH SPECIFIED

         DS    F          RESERVED
NSGLTH   EQU   *-CHANSG   LENGTH OF PARAMETER LIST
```

DSEG must be the object of an execute instruction and be fullword aligned.

Execution: This macro instruction passes control to the resident supervisor module CEAP4 via SVC 182. An attempt will then be made to name and disconnect the specified segment group.

Programming Note: The system programmer should use the DISCSEG macro instruction described in Assembler User Macro Instruction, GC28-2004.

DSSEG -- Disconnect Shared Page Table From Segment (R)

The DSSEG macro instruction disconnects a segment table entry from a shared page table.

Name	Operation	Operand
[symbol]	DSSEG	

Note: There are no operands; see Initialization.

Initialization: The DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding DSSEG. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding DSSEG must be issued with the PRIVILEGED option.
Register 0 must contain a known shared page table number right adjusted in the register. Register 1 must contain a relative page number (left half) and page count (right half) that describes a range of addresses contained in the shared segment group.

Execution: The address space defined by the input parameters is disconnected from the requesting task's virtual storage. If this results in an entire segment or group of segments being free, the segments are disconnected and the virtual storage is made unassigned.

Programming Note: The DSSEG macro instruction is used by the privileged system to keep track of a task's shared page and segment tables. For example, a disconnect request for virtual storage (VMA) causes a DSSEG macro instruction to be issued. The system programmer should use the disconnect entry point (CZCGA8) in VMA to accomplish this function.

Example: Suppose two pages of shared page table number X'FF10' starting at relative page number 3 are to be disconnected from a task's virtual storage. The code might be written

```
         LM   R0,R1,SPTPC
ANY      DSSEG
           .
           .
           .
SPTPC    DC   A(X'0000FF10'),A(X00030002')
```

DUPCLOSE — Close a Duplexed Data Set (S)

Note: this macro must not be used for new code. It is documented here only to aid in the maintenance of existing programs.

The DUPCLOSE macro instruction disconnects the primary and secondary data sets of a duplex data set from the user's problem program and removes the duplexing linkage between such data sets.

Name	Operation	Operand
[symbol]	DUPCLOSE	dcb address 1,dcb address 2[,MF={E|(L,list)}]

Note: A symbol is required in the name field of the L-form. If the MF operand is omitted, the standard form is assumed.

dcb address 1
specifies the address of the data control block opened for the primary data set that is to be permanently disconnected (closed) from the system and removed from duplex data set mode.

Specified as: In the standard and L-form, as a relocatable expression; in the standard and E-form, in register notation (2 through 12); in the E-form only, as an explicit or implied address.

dcb address 2
specifies the address of the data control block opened for the secondary data set that is to be permanently disconnected (closed) from the system and removed from duplex data set mode.

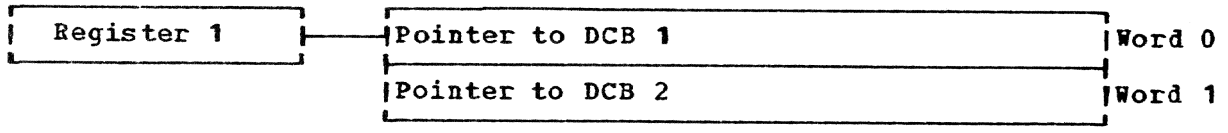
Specified as: In the standard and L-form, as a relocatable expression; in the standard and E-form, in register notation (2 through 12); in the E-form only, as an explicit or implied address.

Initialization: If this macro instruction is to be executed in a privileged module, the most recently issued DCLASS macro instruction in the assembly must have specified PRIVILEGED (see Appendix M). Also, the address of a save area must be placed in register 13 before this macro instruction is executed.

CAUTION: The following errors cause the results indicated:

Error	Action
Permanently closing a duplex data set whose data control blocks are already closed	No action
A dcb address operand does not specify the address of a data control block	Task terminated
DSORG specification is invalid	Task terminated

Programming Notes: The DUPCLOSE macro instruction generates code which places a pointer to a parameter list in register 1.



The DUPCLOSE macro instruction releases any sharing interlocks set for either the primary or secondary data set.

Examples: See the examples under the description of the DUPOPEN macro instruction.

DUPOPEN -- Open Duplex Data Set (S)

> Note: this macro must not be used for new code. It is documented here only to aid in the maintenance of existing programs.

The DUPOPEN macro instruction links a primary and secondary data set together so that all changes to the primary data set are immediately reflected in the secondary data set. Thus, at any instant, the two data sets are exact copies of one another. DUPOPEN is used only with VAM data sets.

Name	Operation	Operand
[symbol]	DUPOPEN	dcb address 1,dcb address 2[,option] [,MF={E|(L,list)}]

Note: A symbol is required in the name field of the L-form. If the MF operand is omitted, the standard form is assumed.

dcb address 1
: is the address of the data control block corresponding to the primary data set.

  Specified as: In the standard and L-form, as a relocatable expression; in the standard and E-form, in register notation (2 through 12); in the E-form only, as an explicit or implied address.

dcb address 2
: is the address of the data control block corresponding to the secondary data set.

  Specified as: In the standard and L-form, as a relocatable expression; in the standard and E-form, in register notation (2 through 12); in the E-form only, as an explicit or implied address.

option
specifies the intended method of input/output processing of the data set being connected to the system.

Specified as: The various processing options and their meanings are:

Code	Meaning
INPUT	The data set can be used as input only.
OUTPUT	The data set can be used for output or input.
INOUT	Both input and output operations are allowed. The DCB is opened as INPUT.
OUTIN	Both output and input operations are allowed. The DCB is opened as OUTPUT.
UPDAT	The data set can be updated.

Note: Opening a VISAM data set for INOUT or OUTIN is equivalent to opening for UPDAT. When a data set is opened for UPDAT, however, the user must position to the desired record in the data set.

Default: INPUT

Initialization: If this macro instruction is to be executed in a privileged module, the most recently issued DCLASS macro instruction in the assembly must have specified PRIVILEGED (see Appendix M). Also, the address of a save area must be placed in register 13 before this macro instruction is executed.

CAUTION: If either copy of a duplexed data set is changed independently of the other, duplexing is invalidated in a manner that is transparent to the duplexing mechanism and may cause false recoveries.

Programming Notes: DUPOPEN should be used judiciously. The external storage required to contain a duplexed data set is exactly doubled and the time required for data output is approximately doubled.

The data set properties or attributes specified in each of the data control blocks and their corresponding DDEF statements must be identical. Similarly, any sharing properties specified by PERMIT commands for each copy of a duplexed data set must be consistent.

In event of an input error on a primary data set page, the corresponding page of the secondary data set is obtained and written back into the primary data set. This not only recovers the error but tends to keep the primary copy in an error-free state.

Each copy of the duplexed data set is placed on a separate physical volume, if possible, thereby further assuring successful error recovery.

The DUPOPEN macro instruction generates the following parameter list:

Register 1	Pointer to DCB 1
	Pointer to DCB 2
	Pointer to option (using same translation as OPEN)

It then generates code which places a pointer to that parameter list in register 1, and links to the DUPOPEN routine via a standard linkage.

Examples: In EX1 the data set defined by DCB named MASTER is opened as the primary data set of a duplex data set and the data set described by the DCB named COPY is opened as the secondary data set. Both data sets are opened for INOUT processing. In EX2, the same two data sets are re-opened, following disconnection caused by DUPCLOSE, for default (INPUT) processing.

```
DDEFIN1   DDEF       MASTDATA,VS,DSNAME=MASTFILE
DDEFIN2   DDEF       COPYDATA,VS,DSNAME=DUPLICAT
MASTER    DCB        DDNAME=MASTDATA,DSORG=VS,RECFM=F,LRECL=80
COPY      DCB        DDNAME=COPYDATA,DSORG=VS,RECFM=F,LRECL=80
EX1       DUPOPEN    MASTER,COPY,INOUT
          DUPCLOSE   MASTER,COPY
EX2       DUPOPEN    MASTER,COPY
          DUPCLOSE   MASTER,COPY
          OPEN       (MASTER,(INPUT))   Subsequent processing after this
                                        instruction would invalidate
                                        previous duplexed data set.
```

ENABLE -- Enable System Interrupts

ENABLE permits the user to "enable" system interrupts and address translation. It sets a new system mask, and if desired by the user, saves the previous system mask in a specified area.

Name	Operation	Operand
[symbol]	ENABLE	[AREA=] [,IO=] [,EXT=] [,PER=] [,TRAN=]

AREA
area in which the current system mask is to be stored

Specified as: an RX address or register notation

Default: Current system mask is stored in a field in the PSA-PSASM

IO=
I/O interrupts

Specified as:
Y - enable I/O interrupts.
N - status of I/O interrupts is unchanged.

Default: Y.

EXT=
external interrupts.

Specified as:
Y - enable external interrupts.
N - status of external interrupts is unchanged.

Default: Y.

PER=
Program Event Recording Interrupts.

Specified as:
Y - enable program event recording interrupts.
N - status of program event recording interrupts is unchanged.

Default: N.

TRAN=
address translation.

Specified as:
Y - enable address translation.
N - status of address translation is unchanged.

Default: N.

Execution: ENABLE uses the S/370 instruction STNSM.

Return Data: AREA, if defined, contains the previous system mask after execution of the ENABLE.

Examples: To enable only I/O interrupts, you may write:

```
DISABLE AREA=AREA

ENABLE IO=Y,EXT=N
```

The DISABLE is done first to insure the rest of the interrupt types are disabled and to save the current system mask. The ENABLE is done to enable only I/O interrupts. To restore the previous system mask, you could write:

```
        SSM SAVE

SAVE DC X'00'
```

ENQGQE -- Enqueue GQE from SCAN Table

ENQGQE sets up the parameter list and calls CEAJEN to enqueue a GQE from the SCAN table.

Name	Operation	Operand
[symbol]	ENQGQE	GQE

GQE
address of the GQE to be enqueued.

Specified as: the name of a symbol defined as a fullword, or Register 2 through 12 or 1.

CAUTIONS: The PSA dsect CHAPSA must have been copied and assigned a base prior to the use of this macro. The expansion of this macro is affected by use of the CVT macro prior to the use of this macro.

Examples:

(1) Register notation; CVT YES is assumed

```
ENQGQE (1)
* L  15,CVTJEN
* BALR 14,15
```

(2) Symbolic name; CVT YES is assumed

```
      ENQGQE TCWGQE          where TCWGQE is defined as a fullword
        .
        .
+   CHDVAL TCWGQE,1
+   L 1,TCWGQE
+   L 15, CVTJEN
+   BALR 14,15
```

(3) Symbolic name; CVT NO is assumed

```
     ENQGQE TCWGQE          where TCWGQE is defined as a fullword
```

```
+    CHDVAL TCWGQE,1
+    L 1,TCWGQE
+    L 15,PSACVT
+    L 15,CVTJEN-CHACVT(,15)
+    BALR 14,15
```

ENTER -- Enter Privileged Service Routine (R)

Name	Operation	Operand
[symbol]	ENTER	

Note: There are no operands. See Initialization.

Initialization: Register 15 must contain the enter code associated with the privileged module being entered by the nonprivileged program executing the ENTER macro instruction. Enter codes are summarized in Appendix A.

Execution: A task-SVC interruption is created to transfer control to the task monitor. The Enter routine (part of the task monitor) transfers control to the indicated privileged program, using modified type-1 linkage. Register 15 contains a code, the enter code, that is used by the Enter routine to determine which privileged program is to receive control. Only the contents of registers 0 and 1 are passed to the privileged program; registers 0, 1, and 15 are the only registers the privileged program can use to pass results back to the program issuing the ENTER. Registers 2 through 14 are saved and restored by the enter routine for the program that issued ENTER.

Example: Suppose you want to get 256 bytes of working storage (without using the GETMAIN macro instruction). You might write:

```
         SR      1,1     CLEAR GP R1 TO SET OPTIONS
         LA      0,256   SET BYTE COUNT
         LA      15,48   SET ENTER CODE IN GP R15
NAME     ENTER
```

ERROR -- Indicate Supervisor Detected Error (O)

The ERROR macro instruction provides the means by which the resident supervisor reports the occurrence of a major or minor software error, a hardware failure, or an I/O error.

Name	Operation	Operand
[symbol]	ERROR	error type,fillin,message id,call id

error type
specifies the type of error that has occurred.

Specified as: One of the codes shown in Figure 24.

Type of Error	Code
Minor software error	1
Major software error	2
Hardware failure	3
Issue message	7

Figure 24. System error codes

fillin
this operand must be included for compatibility.

Specified as: Any two-digit decimal number in the range 00 through 27.

message id
specifies a unique identifier for the message to be issued as a result of invoking the ERROR macro instruction.

Specified as: A two-digit decimal number in the range 01 through 99.

call id
specifies a specific ERROR call in modules that issue more than one call.

Specified as: A decimal number in the range 0 through 99.

Note: Both LVPSW and ERROR use the same SVC code (254); an SVC 254 occurring in the problem state is considered LVPSW; an SVC 254 occurring in the supervisor state is considered ERROR.

Execution: SVC 254, generated by the ERROR macro instruction, is the only SVC that may be issued by the resident supervisor. The processing unit receiving the SVC will stop all other processing units in the system. A message (for its format, see "Syser Dump" in Part I, Section 4) is issued at the operator's terminal and the system enters the wait state; a dump may then be taken.

If the error type is 1 (minor software), or if the recovery nucleus returns control to the SVC 254 routine, all other processing units in the system are restarted; control is then returned to the instruction following the ERROR parameter list. If the error type is 2 (major software), the system is suspect, and the IPL process must be initiated by the operator. If the error type is 3 (hardware failure), the error is handled as a type-2, but the word SOFTWARE in the message is changed to HARDWARE. If the error type is 7, the system is stopped, a message is immediately written to the operator console, and the system is restarted. See the example below to learn how to set up the message. This is primarily used for an I/O failure, to inform the operator of some action to take concerning the failing device.

Example: To send a message to the operator, you might code:

```
SENDMESS   ERROR  7,0,0,0
           ORG    *-3
           DC     AL3(MESSTEXT)
           .
           .
           .
MESSTEXT   DC     AL1(4)          MESSAGE LENGTH
           DC     C'HELP'
```

Programming Note: Part of the message issued at the operator's terminal is a four-digit error code; this code is formatted from the two-digit message id and the two-digit call id passed to the system error processor as parameters of the ERROR macro instruction. This construction permits you to identify calls to the system error processor for debugging. You might, for example, assign one module or one group of modules in main storage a particular module identifier to permit its recognition as the source of the call. You could then, using the call id operand, assign sequential numbers to the ERROR calls issued by that module or group of modules to help you recognize particular errors resulting in calls within the sequence. For example, you might write:

```
                        ┌──────message id
                        │  ┌───call id
ERROR     1,00,23,10
```

and the resulting error code, 2310, would identify the error that resulted in the call to the system error processor.

To avoid issuing different ERROR calls with the same error code (thus duplicating the messages issued at the operator's terminal and creating confusion as to the reason for the call), check System Messages for those error codes already in use in the system.

Example: Suppose you detect a major error -- quantity A was neither less than, equal to, nor greater than quantity B. You might write:

```
|       BLAST   ERROR   3,01,23,02
```

ESEG -- Exchange Named Segment (0)

The ESEG macro instruction transfers control to the exchange named segment program in the resident supervisor. An attempt will then be made to exchange the specified disconnected segment group with the virtual storage segment specified.

Name	Operation	Operand
	ESEG	

Note: There are no operands.

Initialization: Before executing ESEG , the issuing program should have set up the following parameter area:

```
|     CHANSG   DSECT ,        COMMON NAMESEG PARAMETER LIST
             DS    0F
      NSGSVC   DS    H        SVC
|     NSGFLG   DS    X        FLAGS
      NSGGET   EQU   NSGFLG
|     NSGGETM  EQU   B'80'    GET DISCONNECTED SEGMENT PAGE
      NSGPUT   EQU   NSGFLG
|     NSGPUTM  EQU   X'40'    PUT DISCONNECTED SEGMENT PAGE
             DS    X        RESERVED

|     NSGRNA   DS    XL8      RESERVED SEGMENT GROUP NAME
|     NSGDNA   DS    XL8      DISCONNECTED SEGMENT GROUP NAME

|     NSGVMA   DS    A        VIRTUAL STORAGE ADDRESS OF SEG GROUP

|     NSGLNG   DS    H        LENGTH OF NAMED GROUP

|     NSGFLI   DS    XL1      INPUT FLAGS
```

```
NSGFLO   DS    XL1      OUTPUT FLAGS
NSGDNGM  EQU   X'80'    DNAME SPECIFIED
NSGRNGM  EQU   X'40'    RNAME SPECIFIED
NSGADGM  EQU   X'20'    ADDRESS SPECIFIED
NSGBNDM  EQU   X'10'    MODE=BOUND
NSGLNGM  EQU   X'08'    LENGTH SPECIFIED
NSGRELM  EQU   X'04'    RELEAS=Y SPECIFIED
NSGRESSM EQU   X'02'    RSTRCT=Y SPECIFIED

NSGBVA   DS    F        GET/PUT SEGMENT BUFFER ADDRESS
NSGLTH   EQU   *-CHANSG  LENGTH OF PARAMETER LIST
```

ESEG must be the object of an execute instruction and be fullword aligned.

Execution: This macro instruction passes control to the resident supervisor module CEAP8 via SVC 185. An attempt will then be made to exchange the specified disconnected segment group.

Programming Note: ESEG is an inner macro used with the EXCSEG macro instruction.

EXCSEG -- Exchange Segment Group (O)

This macro instruction is completely documented in the Assembler User Macro Instruction manual, except for one operand that is available only to a systems programmer. The definition and specification for only that one operand are given below, but for continuity, the metalanguage format that follows shows all the operands.

L-form

Name	Operation	Operand
Symbol	EXCSEG	[DNAME=,LENGTH=,BOUND=,RNAME=,RELEAS=,] MF=L

E-form

Name	Operation	Operand
[symbol]	EXCSEG	[DNAME=,LENGTH=,BOUND=,RNAME=,ADDRESS=,RELEAS=,] MF=(E,LIST)

Standard-Form

Name	Operation	Operand
[symbol]	EXCSEG	DNAME=[, LENGTH=,BOUND=,RNAME=,ADDRESS=,RELEAS=,]

Note: all operands are keyword.

RELEAS=
specifies whether or not the reserved segment GETMAIN restriction is enforced while the named segment group is disconnected.

Specified as:

Y - the reserved segment GETMAIN restriction is not enforced.

N - the reserved segment GETMAIN restriction is enforced.

Default: N

This option is available to privileged class programs only. Refer to GETMAIN and RSVSEG macros for further explanation.

EXPND -- Expand Page (O)

The EXPND macro instruction transfers control to the Expand Page Program (Ceap1) in the Resident Supervisor. An attempt will then be made to allocate additional virtual memory address space contiguously to existing virtual memory address space.

Name	Operation	Operand
[symbol]	EXPND	

Note: There are no operands.

Initialization: Before executing EXPND, the issuing program should have set up the following general registers:

Register	Contents
0	existing number of pages (left half) additional number of pages (right half)
1	starting virtual memory address of existing space
15	protection code for additional address space (see ADDPG macro description)

CAUTION: The existing virtual memory address space must be currently allocated.

Execution: This macro instruction passes control to the Resident Supervisor Module CEAP1 via SVC 239. An attempt will then be made to allocate the specified number of additional virtual memory pages contiguously to the existing address space.

Return Data:

Register	Contents	Meaning
1	virtual memory address of first page allocated	---
15	0	allocation ok
	4	request not satisfied

Programming Note: The system programmer should use the expand entry point (CZCGA4) in VMA to expand virtual memory.

Example: Suppose two additional pages of privileged virtual memory are needed contiguous to the virtual memory address contained in general register 4. The following instructions may be written:

```
         .
         .
         .
         L    0,=A(X'00010002')
         LR   1,4
         LA   15,2
         EXPND
```

```
LTR 15,15
BNZ NOTALLOC
.
.
.
```

FINISH (MSAM) -- End of Data Set (R)

The FINISH macro instruction is used to inform the MSAM routines that processing of the current data group (a subsection of an MSAM data set; a complete data set to the user of the unit record device) is at an end. A task may process more than one data group (within the same MSAM data set) with the same data control block, without closing and reopening the DCB (and the assignment and release of the associated I/O device) between data groups.

Name	Operation	Operand
[symbol]	FINISH	dcb address

dcb address
specifies the address of the data control block opened for the data set being processed.

Specified as: An RX address, or as register notation. Execution time may be saved if register 1 is specified. If register notation is used, the address must first be loaded into the specified register.

CAUTION: The FINISH macro instruction causes the operator to be notified to remove the data group from the device in use if the INHMSG flag of the data ccntrol block is not set to 1.

Return Data: Upon completion of the FINISH macro instruction, a code indicating the manner in which the instruction was completed is returned in register 15. The meanings of the codes are given in Figure 25.

Return Code	Meaning
0	Operation completed successfully. CLOSE macro instruction may be issued, or further processing may be initiated without reopening data control block; the DCB parameters LRECL, MODE, STACK, PRTSP, RECFM, POCKET, FORMTYPE, and RETRY may be altered at this time.
4	Operation was not completed since I/O was not finished. FINISH macro instruction should be reissued later, until a return code other than 4 is received. (See discussion of "Interruption Entry Handling," above.)
8	Operation was completed with I/O error. If data control block was opened for input, description of GET macro instruction return code of 8 (Figure 26) applies; if data control block was opened for output, description of PUT macro instruction return code of 8 (Figure 34) applies. In order to clear remaining output buffers, if error was not permanent (DEBNF2 or DECG1 not on), FINISH may be reissued.

Figure 25. Return codes for MSAM FINISH macro instruction

Programming Notes: The FINISH macro instruction provides for:

- Initiating any necessary I/O activity so that an output data set may be closed.
- Testing the results of all outstanding I/O on an output data set.
- Awaiting completion of outstanding I/O requests on an input data set.
- Notifying an operator to remove the data set from the device (under control of the INHMSG flag of the data control block).
- Reading a card from the card reader and stacking it in pocket 3 of the same 2540 as the selected punch, if the COMBINE flag of the data control block is set.

You should precede the CLOSE macro instruction with the FINISH macro instruction, if you want to avoid an automatic wait condition, which may result from the access method CLOSE, or to receive notification of any I/O error. (Close MSAM is the only MSAM routine to invoke AWAIT.)

Example: The following example is based upon the coding in the example for the PUT macro instruction. It would follow the locate-mode PUT, L, BR:

```
            LA      7,FINTAB            SET UP BRANCH TABLE
REPEAT      FINISH  JHL                 END OF DATA SET
            L       5,0(15,7)           BRANCH ON RC INDEX AND
            BR      5                   FINTAB AS BASE
HALT        CLOSE   JHL
            .
            .
            .
FINTAB      DC      A(HALT)             ADDR FOR PROCESSING AFTER
                                        RC OF 0
            DC      A(REPEAT)           ADDR FOR PROCESSING AFTER
                                        RC OF 4
            DC      A(ERROR)            ADDR FOR PROCESSING AFTER
                                        RC OF 8
```

Return Code	Meaning
0	Operation completed successfully.
4	I/O not complete; no record has been provided since next sequential buffer has not yet been filled; GET macro instruction should be reissued.
8	Unrecoverable I/O error occurred in connection with record being read; normally, CLOSE macro instruction should be issued; however, a record has been provided, content of which is buffer image. If I/O error was not permanent (DEBNF2 or DECG1 not on), you may accept record and continue processing, or you may skip record by issuing another GET macro instruction.
12	End-of-file; no record has been provided. The FINISH macro instruction should be issued.
16	Control card sensed: attempt to read an EBCDIC record resulted in validity check; first four columns of card contain same predetermined control mark, a 12-11-3-4 punch. Record provided is buffer image, control bytes of which should be tested for such installation-defined codes as (1) change of mode from EBCDIC to column binary or (2) change of data group without end-of-file indicator. Depending on installation assignment and control code usage, processing may continue. Control card will be stacked as if it were valid data card. If subsequent GET macro instruction is issued, it will refer to next card in reader following control card. If any fields in data control block are to be changed, FINISH (or CLOSE and OPEN) macro instructions must be issued before the next GET.

Figure 26. Return codes for MSAM GET macro instruction

FREELOCK -- Open a Resident Supervisor Service (O)

The FREELOCK macro instruction opens a specified resident supervisor service that is currently locked, thus allowing the service to be used by another processor.

Name	Operation	Operand
[symbol]	FREELOCK	lock area,register,module id,error number

lock area
: specifies the address of a double word (8 bytes, fullword aligned) that is a currently locked service lock area.

  Specified as: a symbolic address.

register
: specifies a register to be used for logging the FREELOCK address.

  Specified as: an absolute expression from 0 to 15.

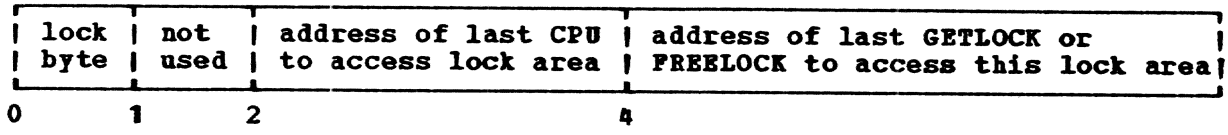
module id
: specifies the module opening the resident supervisor service (the module in which the FREELOCK is issued). This operand will be used in generating a SYSER message in the event the lock area is currently free or locked by some other CPU.

Specified as: a two-digit decimal number.

error number
specifies this particular FREELOCK macro instruction within a module in which more than one SYSER is issued. This operand is used in generating a SYSER message in the event the lock area is currently free or locked by some other CPU.

Specified as: a two-digit decimal number.

Programming Note: The double word service lock area has the following structure:



GET (MSAM) -- Get a Record (R)

The GET macro instruction may be specified in either the locate mode or the move mode. When you specify the macro instruction in the locate mode, GET locates the next sequential record in the specified input data set and places its address in register 1. When you specify the macro instruction in the move mode, GET locates the next sequential record in the specified input data set and moves it to the work area you have specified in virtual storage.

Name	Operation	Operand
[symbol]	GET	dcb address[,area]

dcb address
specifies the address of the data control block opened for the data set being processed.

Specified as: An RX address, or register notation (1 through 12). Execution time is saved if register 1 is specified. If register notation is used, the address must first be loaded into the specified register.

area
specifies the address of the work area into which you want the record to be moved. Use of this operand implicitly specifies move mode.

Specified as: An RX address, or register notation (0 or 2 through 12). Execution time is saved if register 0 is specified. If register notation is used, the address must first be loaded into the specified register.

Initialization: The address of a save area must be placed in register 13 before executing the GET macro instruction.

When you are using MSAM, the GET macro instruction may only be employed to obtain records from a card reader. The RECFM field of the data control block must therefore indicate format-F, since format-V does not apply to the card reader. At OPEN, the LRECL field of the data control block should be set to a maximum of 80 bytes for EBCDIC, or 160

bytes for column binary. The mode field in the data control block must be set to a binary 0 for EBCDIC or to binary 1 for column binary.

Return Data: Upon completion of the GET macro instruction, a code indicating the manner in which the instruction was completed is returned in register 15. Meanings of the codes are given in Figure 26.

Example: In the following example, which illustrates the use of both the locate-mode and move-mode GET macro instructions, you want to read 65 EBCDIC bytes from the first 65 columns of the next sequential card, in a 2540 reader. Any cards with errors will be stacked in bin 2, with no attempt to reread the record; cards containing no errors will be stacked in bin 1. Since the return codes provided from the macro instruction are multiples of 4, it is possible for you to set up a branch table to provide proper control of processing.

```
ADL         DCB   DSORG=MS,MACRF=G,          BUILD DCB
                  DDNAME=MYDD,DEVD=RD,
                  MODE=E,POCKET=2,
                  STACK=1,RECFM=F,
                  LRECL=65
            OPEN  (ADL,(INPUT))              OPEN DCB
            LA    3,RCTABLE                  SET UP BRANCH TABLE
            LA    1,ADL                      LOAD ADDR OF DCB
MOVE        GET   (1),WORK                   MOVE-MODE GET MACRO
            L     5,0(15,3)                  BRANCH ON RC INDEX AND RCTABLE
                                             AS BASE
            BR    5
LOCATE      GET   ADL                        LOCATE-MODE GET MACRO
            L     5,0(15,3)                  BRANCH ON RC INDEX AND
            BR    5                          RC TABLE AS BASE
            .
            .
            .
WORK        DS    CL65                       INPUT AREA FOR MOVE-MODE
                                             GET MACRO
RCTABLE     DC    A(NORM)                    ADDR FOR PROCESSING AFTER
                                             RC OF 0
            DC    A(PAUSE)                   ADDR FOR PROCESSING AFTER
                                             RC OF 4
            DC    A(ERROR)                   ADDR FOR PROCESSING AFTER
                                             RC OF 8
            DC    A(END)                     ADDR FOR PROCESSING AFTER
                                             RC OF 12
            DC    A(CONTROL)                 ADDR FOR PROCESSING AFTER
                                             RC OF 16
```

Both the move-mode and locate-mode GET macro instructions result in type-1 linkage to the DOMSAM routine.

GETADDR -- Get System Address from CVT

GETADDR loads the specified register with the address of the specified symbol from the CVT control block.

Name	Operation	Operand
[symbol]	GETADDR	[EPT=] [,R=]

EPT=
name of the system symbol whose address is needed.

R=
the register into which the address of the symbol (specified in EPT=) is to be placed.

Specified as: Any register, except Register 0 cannot be used if CVT NO was coded.

Default: Register 15

CAUTIONS: The PSA dsect CHAPSA must be copied in the assembly using GETADDR. The expansion of GETADDR is affected by the use of the CVT macro prior to the use of GETADDR.

Currently, only those system symbols starting with the following three letters have been placed in the CVT:
CEA
CHB
SCH
CIP

Example:

(1) To get the address of entry point CEATA1 and place it in Register 5 (assuming CVT YES), you might write:

```
  GETADDR EPT=CEATA1,R=R5
+ L R5,CVTTA1
```

(2) To get the address of the core block header in Register 10 (assuming CVT NO) you might write:

```
  GETADDR EPT=CHBCBH,R=R10
+ L R10,PSACVT-CHAPSA(,0)
+ L R10,CVTCBH-CHACVT(,R10)
```

GETCORE -- Allocation Supervisor Storage Space

GETCORE calls CEAL1A (Supervisor Allocation) to request allocation fo a given amount of storage. Storage is allocated in blocks of 64 bytes with a maximum length of 4096 bytes.

Name	Operation	Operand
[symbol]	GETCORE	[LGH] [,TYPE=] [,RETURN=] [,ERROR=]

LGH
the amount of storage to be allocated.

Specified as: any number greater than 0 and less than or equal to 4096. May be a register (0 is allowed), halfword, fullword, self-defining expression or an EQU.

TYPE=
amount of time the storage space is to be in use.

Specified as: S, M, or L.

S/SHORT - core is to be in use a short time only -- about one or two cycles through the supervisor.
M/MEDIUM - core is to be in use longer than a supervisor cycle or two, but less than the possible life of a task. (Most I/O falls within this time length.)

L/LONG - core is to be in use for a long time, or the life of the task. Examples are TSIs, TSDL page, TCTs, shared page tables, etc.

Default: S.

RETURN=
indicates to system whether or not to return if sufficient storage is not available to fulfill request.

Specified as:
Y - return if sufficient storage is not available; ERROR (see below) must also be coded.
N - do not return if sufficient storage is not available.

Default: N.

ERROR=
label of the 'branch to' if storage space cannot be allocated; valid only if RETURN=Y.

CAUTIONS:
(1) The PSA dsect CHAPSA must be copied in the assembly containing the GETCORE macro.
(2) The expansion fo this macro is affected by the use of the CVT macro prior to the use of this macro.
(3) If the system is unable to fulfill the storage request and 'RETURN=N', then the system will be terminated with the major syserr 571.

Return Data:

Length of the storage area allocated is in Register 0.
Address of the storage area allocated is in Register 1.

Examples:

(1) Using self-defining term for length of request

```
        GETCORE 64,TYPE=L
```

results in 64 bytes being allocated from the long term storage chain.

(2) Using a symbol pointing to the length, and return requested

```
        GETCORE LGH1,TYPE=M,RETURN=Y,ERROR=NOCORE

NOCORE DS 0H
*
*       recover; sufficient storage not available
*
LGH1   DC F'1024'
```

results in 1024 bytes of storage being allocated. If unable to allocate, CEAL1A will return to caller and the GETCORE macro will automatically branch to the NOCORE label.

GETLOCK -- Lock a Resident Supervisor Service (O)

The GETLOCK macro instruction locks a specified resident supervisor service, thus serializing the service in a multi-processor environment.

Name	Operation	Operand
[symbol]	GETLOCK	lock area,register,[module id],[error number], {IMMEDIATE|WAIT},[action],[exit address]

lock area
specifies the address of a double word (8 bytes, fullword aligned) that is the service lock area.

Specified as: a symbolic address.

register
specifies a register to be used for counting time increments and for logging the SETLOCK address.

Specified as: an absolute expression from 0 to 15.

module id
specifies the module trying to lock the resident supervisor service (the module in which the getlock is issued). This operand will be used in generating a SYSER message in the event the attempt to set the lock byte is unsuccessful. This operand is required if the WAIT operand is specified; it is ignored it the immediate operand is specified.

Specified as: a two-digit decimal number.

error number
specifies this particular GETLOCK macro instruction within a module in which more than one SYSER is issued. This operand is used in generating a SYSER message in the event the attempt to set the lock byte is unsuccessful. This operand is required if the WAIT operand is specified; it will be ignored if the IMMEDIATE operand is specified.

Specified as: a two-digit decimal number.

IMMEDIATE|WAIT
specifies how long to keep trying to set the lock in the event it is already locked.

Specified as:
IMMEDIATE -- only try to set it once
WAIT -- keep trying until a length of time has elapsed

action
specifies what to do if the attempt to set the lock fails.

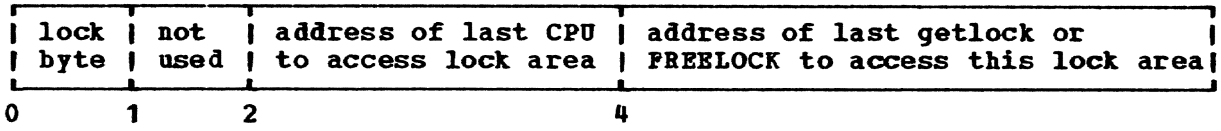
Specified as: a symbolic address to which to branch if the IMMEDIATE operand was specified.

exit address
specifies where to branch to if the lock was set or if a SYSER occurred.

Specified as: a symbolic address.

Default: The program continues at the next sequential instruction.

Programming Note: The double word service lock area has the following structure:



GETPAG -- Get Virtual Memory Page

GETPAG sets up a parameter list and calls CEATC7 to request all pages, defined by a virtual memory address and length, be brought into storage and placed in page hold.

Name	Operation	Operand
[symbol]	GETPAG	TSI=,TCW=

TSI=
address of the locked TSI for the owner of the pages.

Specified as: an RX address or register notation

TCW=
address of a parameter list for CEATC7 containing the starting virtual memory address and length of the area needed to be held in storage.

Specified as: an RX address or register notation

CAUTIONS: The PSA dsect CHAPSA must be copied into the assembly containing the GETPAG macro. The expansion of this macro is affected by the use of the CVT macro prior to the use of this macro.

programming Note: The dsect CHATCW describes the TCW parameter list area.

Execution: Using the parameter list specified by TCW and described by the CHATCW dsect, CEATC7 builds a parameter list for CEAMQ and makes a type 'C' call to CEAMQ to read the needed pages and place them in page hold. On return from CEAMQ, CEATC7 fills in the associated real core addresses and lengths in the TCW parameter list.

Return Codes: The return codes in Register 15 are:

Code	Meaning
0	successful request
negative	paging error; exit
4	invalid VMA given

GETWORK -- Get Temporary Work Area

The GETWORK macro allocates a temporary work area of specified length from the supervisor save area stack.

Name	Operation	Operand
[symbol]	GETWORK	LGH [,BASE=]

LGH
length of work area to be allocated.

Specified as: a number that is a multiple of 4, or a symbol that is defined on a fullword boundary. If given as a number that is not a multiple of four, this macro rounds the number to the next higher multiple of four. If given as a symbol, it is the programmer's responsibility to make sure the symbol is defined as a fullword to maintain a fullword boundary.

BASE=
the register in which the work area address is to be returned.

Specified as: Register 1 through 15; 0 is not allowed.

Default: Register 1

CAUTIONS:
(1) The PSA dsect CHAPSA must have been copied prior to the use of this macro.
(2) The GETWORK macro should not be used in a loop.

Programming Notes: There is no macro for freeing space allocated by the GETWORK macro. When the module exits, the save area stack is "popped" which automatically frees the work area.

Examples:

(1) To allocate a temporary 8-byte work area, code:

```
GETWORK 8
```

Register 1 upon exit from the macro will contain the work area address.

(2) To allocate a temporary work area and have the address returned in a register other than register 1, code:

```
GETWORK 64,BASE=12
```

Register 12 upon exit from the macro will contain the address of the 64-byte work area.

GNC -- Get Next Character (O)

The GNC macro gets the next character from the command system's source list, tests it for a specific function character, and if not a specific function character, makes it available to the macro user.

Name	Operation	Operand
label	GNC	[exit] [,attn]

exit
the label of the branch-to if the source list is to be refreshed because the end of the current level has been reached.

Specified as: an RX address.

Default: get the next character.

attn
the label of the branch-to if the source list handler detects an attention from the user while processing the source list.

Specified as: an RX address.

Default: ignore, and get the next character.

Programming Note: The GNC macro uses registers 1, 14 and 15. The macro expects the user to have issued a USING and a COPY for the dsect CHASLH. Also, the user must copy the dsects CHANTC and CHAPCT. The GNC macro does a USING for CHANTC using register 15 and a drop on register 15. The text character is returned to the user in register 1.

Execution: GNC retrieves the character pointed to by the address in SLHCSA. The character is tested for an 'EOB' (X'26'). If it is and the following character is not an 'E' CZASC3 is called to process the source list marker. On return from CZASC3, the return code is tested for possible exit conditions. If no exit conditions exist or if an exit or attn label were not given, GNC attempts to retrieve the next source list character. If the following character is an 'E' the macro exits to the user with the EOB character in register 1.

If the retrieved character is not an EOB, a test is made to determine if the character is a continuation character by testing against the user's appropriately-defined character in the Profile Character Table (CHAPCT). If the character is not a continuation character, GNC exits to the user after updating SLHCSA to point to the next character. If the retrieved character is a continuation and is followed by an 'EOB,E' sequence (X'26C5), the SLHCSA is updated to point to the EOB,E sequence and CZASC4 is called to process the E marker (i.e., the 'EOB,E' sequence). On return, the return code is tested for a possible exit condition, if exit or attn were specified. If not specified or the exit condition is not satisfied, the next character from the source list is retrieved.

GPSEG -- GET/PUT Named Segment (0)

The GPSEG macro instruction transfers control to the GET/PUT named segment program in the resident supervisor. An attempt will then be made to get a page from or put a page into the specified disconnected segment group.

Name	Operation	Operand
	GPSEG	

Note: There are no operands.

Initialization: Before executing GPSEG, the issuing program should have set up the following parameter area:

```
         CHANSG   DSECT ,          COMMON NAMESEG PARAMETER LIST
                  DS    0F
         NSGSVC   DS    H          SVC
         NSGFLG   DS    X          FLAGS
         NSGGET   EQU   NSGFLG
         NSGGETH  EQU   N'80'      GET DISCONNECTED SEGMENT PAGE
         NSGPUT   EQU   NSGFLG
         NSGPUTH  EQU   X'40'      PUT DISCONNECTED SEGMENT PAGE
                  DS    X          RESERVED

         NSGRNA   DS    XL8        RESERVED SEGMENT GROUP NAME
         NSGDNA   DS    XL8        DISCONNECTED SEGMENT GROUP NAME

         NSGVMA   DS    A          VIRTUAL STORAGE ADDRESS OF SEG GROUP
```

```
|       NSGLNG   DS    H         LENGTH OF NAMED GROUP

|       NSGFLI   DS    XL1       INPUT FLAGS
|       NSGFLO   DS    XL1       OUTPUT FLAGS
|       NSGDNGM  EQU   X'80'     DNAME SPECIFIED
|       NSGRNGM  EQU   X'40'     RNAME SPECIFIED
|       NSGADGM  EQU   X'20'     ADDRESS SPECIFIED
|       NSGBNDM  EQU   X'10'     MODE=BOUND
|       NSGLNGM  EQU   X'08'     LENGTH SPECIFIED
|       NSGRELM  EQU   X'04'     RELEAS=Y SPECIFIED
|       NSGRESSM EQU   X'02'     RSTRCT=Y SPECIFIED

|       NSGBVA   DS    F         GET/PUT SEGMENT BUFFER ADDRESS
|       NSGLTH   EQU   *-CHANSG   LENGTH OF PARAMETER LIST
```

| GPSEG must be the object of an execute instruction and be fullword
| aligned.

| Execution: This macro instruction passes control to the resident super-
| visor module CEAQ8 via SVC 186 . An attempt will then be made to get a
| page from or put a page into the specified disconnected segment group.

HOOK -- Transfer Control from IVM to Private Module (O)

The HOOK macro instruction, when placed at each executable entry point of a new IVM module, sets up the mechanism to transfer control from the system version of the module to a private version of the same module.

Name	Operation	Operand
[symbol]	HOOK	[NL]

NL
: indicates that the user does not want the symbol specified in the name parameter to be generated on the first executable instruction of the macro expansion.

  Specified as: NL, or it is left blank.

  Default: The symbol, if specified in the name parameter, is generated.

Programming Note: The most important part of the 'hook' process is the existence of the HOOK macro at every executable entry point in the IVM module.

ID -- Define BULKIO Module ID

The ID macro instruction defines a variable global identifier (BULKIO section ID). This identifier is used by other macros unique to the BULKIO modules.

Name	Operation	Operand
	ID	module id

module id
the variable global identifier.

Specified as: an alphabetic character from A to Z, corresponding to the last character of a BULKIO module CZAW(A/Z).

INVOKE -- Transfer Control (O)

INVOKE transfers control from one program or routine to another by means of the BASR instruction.

Name	Operation	Operand
[symbol]	INVOKE	address

address
specifies the address of a word that contains the address of the program to be invoked.

Specified as: An RX address, or as register notation. If register notation is used, the address must first be loaded into the specified register.

Execution: The specified address is placed into register 15 and a BASR 14,15 is generated.

IOCAL -- I/O Call (E)

The IOCAL macro instruction provides for the initiation of an I/O operation.

Name	Operation	Operand
[symbol]	IOCAL	

Note: There are no operands.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a module prior to coding IOCAL. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding IOCAL must be issued with the PRIVILEGED option.

Execution: An IOCAL macro instruction must always be the subject of an Execute instruction. The supervisor call instruction (SVC 231) is assumed to occupy the first halfword of a variable-length parameter list called an I/O request control block (IORCB). The IORCB supplies the information needed by the supervisor to perform the requested I/O operation. This control block is described by the DSECT CHAIOR.

An IORCB consists of four parts: a fixed part of 10 doublewords; an optional I/O data buffer which cannot exceed 225 doublewords; an optional page list which cannot exceed eight doublewords; and a channel command word (CCW) list. The entire IORCB cannot exceed 240 doublewords and must be contained within a single page.

When an IOCAL is executed, the supervisor, after some error checking, obtains main storage for the IORCB and copies it into that space. Based on the options selected by the user and indicated in the IORCB, the supervisor obtains a path to the requested I/O device, translates the virtual CCW addresses to real storage addresses, brings any required buffer pages into main storage, and starts the I/O operation. After the I/O operation is complete, the supervisor releases the device path, allows the data buffer pages to be paged out of storage if necessary and queues a pending task-I/O interruption for the task associated with the IORCB.

When the pending interruption is accepted by the task, the supervisor copies the IORCB into the task's interruption storage area (ISA). After the IORCB has been copied, the main storage it required is released and the IOCAL operation is completed. A task may have more than one IOCAL in operation at one time and may operate asynchronously with an active IOCAL.

Figure 27 shows the format of the fixed area of the IORCB as it is before an IOCAL. Every IOCAL must have a 20-word fixed area regardless of the fields used. You may use space within the IORCB itself as a data buffer; you can do this if the data does not exceed 225 doublewords (or whatever space is left in the IORCB after the other items you need are included). You must include an IORCB data buffer if you are using a direct access device and if you wish record zero to be read into the IORCB by the supervisor if a unit check occurs. In this case, the IORCB data buffer (the first 56 bytes) is used to hold record zero. You may also use data buffers outside the IORCB; if you do this, you must include a page list. The page list contains one doubleword entry for each virtual storage buffer page; you may not have more than eight page-list entries. Figure 28 shows the organization of a page-list entry.

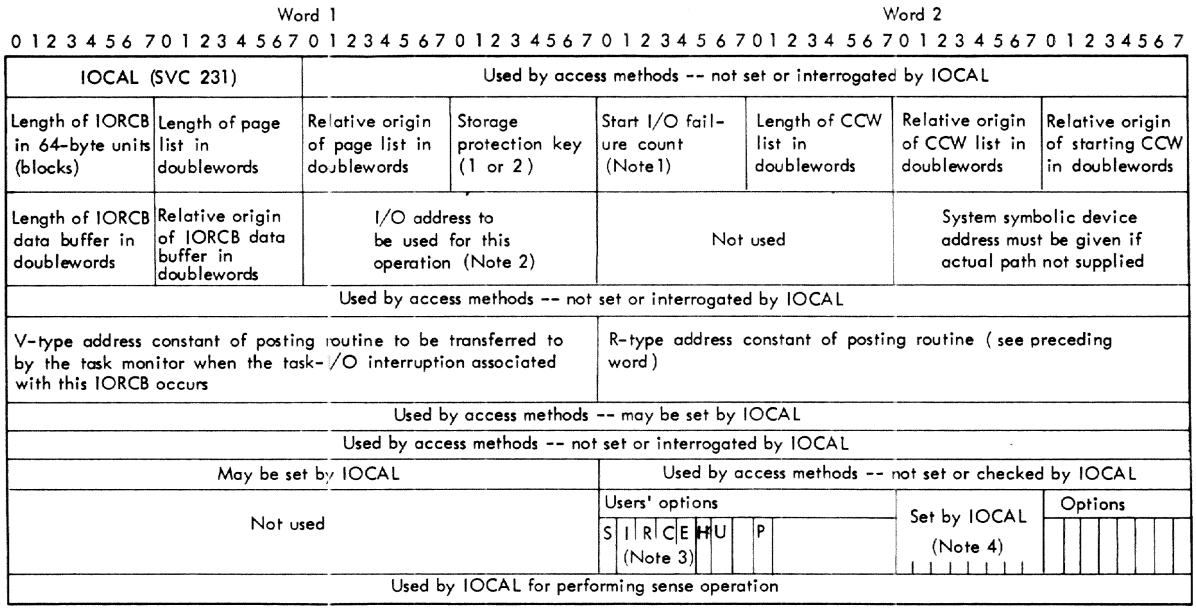
Each page-list entry is associated with a channel command word (CCW) list entry; the CCW entry tells what operation is to be performed with the data buffer. A CCW entry does not need to point to a page-list entry; a page-list pointer of zero is assumed to mean that the IORCB buffer is to be used. You can use any number of CCW entries as long as the size limit of the IORCB is not exceeded. Figure 29 illustrates the format of a CCW list entry. You always have at least one CCW entry, since the CCW represents the work you want the supervisor to do.

If the software command chain flag is 1 (see Figure 30), the supervisor continues to reissue SIO (Start I/O) instructions at the current point in the CCW list when a device-end interruption is received. This has the effect of making the CCW list appear chained, even though the path to the I/O device may be free for certain periods during the operation. This command chaining terminates when all CCWs in the IORCB have been processed. The most common use of software command chaining is to use a standalone seek to position the disk arm followed by a channel program to read or write data. This command chaining terminates when all CCWs in the IORCB have been processed.

If the IORCB chain flag is 1 (see Figure 27), the supervisor changes the last CCW entry to a transfer in channel (TIC) command if another (second) IOCAL for the same device is received before the final channel-end/device-end interruption for the first IORCB is received by the supervisor. This TIC command links the CCW lists of the two IORCBs. The supervisor will also set the program controlled interruption (PCI) bit on, in the start CCW of the second IORCB. The receipt of this PCI signals the completion of activity for the first IORCB; the supervisor then queues a pending task-I/O interruption for that IORCB.

The IORCB received by the task monitor as a result of the task-I/O interruption has been changed by the supervisor; it is not identical to the IORCB originally received by the supervisor. Figure 30 shows the fields in the fixed area of the IORCB that may be set by the supervisor. Figure 31 shows the changes to the CCW list entry.

As part of its interrupt-handling logic, the task monitor transfers control to the posting routine pointed to by the IORCB it receives as a by-product of the task-I/O interruption. This posting routine informs the program originally issuing the IOCAL that the I/O operation has been completed.



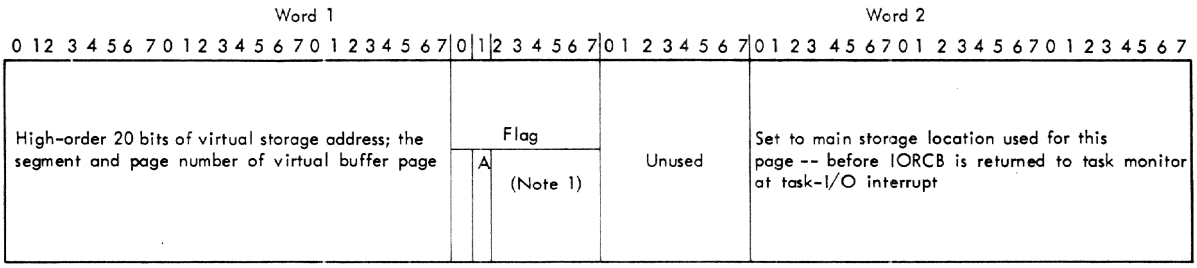
Note 1. If flag R (Note 3) is one, the Start I/O instruction is reissued the number of times specified by this count, or until the Start I/O instruction is successfully initiated.

Note 2. If flag S (Note 3) is one, the I/O address contained in this halfword is used and the symbolic device address is ignored.

Note 3. S=specific I/O address, I=ignore device malfunctioning indicator in pathfinder; R=if Start I/O not accepted because device is busy, reissue Start I/O (see Note 1); C = Software command chain; E = error retry mask; H=issue Halt I/O on device before start I/O; U=if unit check occurs, read direct access device record zero into IORCB data buffer; P=treat PCI as channel end/device end.

Note 4. IORCB chaining flag; if another IOCAL is received for this device while the channel program for this IORCB is running, the last CCW of this list is chained to the first CCW of the other IORCB.

Figure 27. Format of fixed area of input/output request control block as set before IOCAL



Note 1. A = (paging storage) copy of this page does not need to be used; use any main storage page and release paged copy

Figure 28. Organization of a page list entry

Word 1				Word 2		
0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3	4 5 6 7 0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7 0	1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
CCW operation code	Position of page list entry 1,2...8 (Note 1)	Flags 1 (Note 2)	Displacement within page buffer or from start of IORCB buffer or from start of CCW list if TIC	CCW flags	Zeros	Byte count

Note 1. If this field is 0, the IORCB data buffer is assumed

Note 2. 1 = do not relocate CCW addresses

Figure 29. Channel command word list entry before IOCAL is issued

Word 1				Word 2		
0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7 0	1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
Unchanged						
Unchanged						
Unchanged		Set to actual I/O address used for this operation, or left unchanged if user supplied		Unchanged		
Unchanged						
Unchanged						
Unchanged				Condition Codes 1 1 1 I C C S S (Note 1)		Main storage address used for IORCB data buffer. If no IORCB buffer used set to end of fixed area in IORCB; buffer must follow fixed area.
Unchanged						
Sense condition codes 1 1 C C S S (Note 2)	Sense CSW status placed here if both requested operation and sense operation fail	Sense failed fg 0 (Note 3)	Halt I/O retry count (Note 4)	Unchanged		
Unchanged				Flags S P I H R N W T x x C (Note 5)		Not used
CCW for performing sense operation on requested I/O device						

Note 1. If operation came to abnormal end, these condition codes are stored: I=test I/O condition code; C=test channel condition code; S=start I/O or halt I/O condition code.

Note 2. If sense operation failed, these condition codes are stored for I/O instructions used to attempt sense (see Note 1).

Note 3. 0=a device other than the one requested has monopolized the control unit; sense data applies to that device.

Note 4. If user requested both a retry of start I/O and halt I/Os before each start I/O, this field is set equal to the user-supplied start I/O count.

Note 5. S=CCW specification error; P=no path exists to requested device; I=start I/O failed; H=halt I/O failed; R=read record 0 (on direct access error) failed; N=sense failed; W=CCW addresses are relocated (changed to real addr) T=IORCB aborted because previous (pending) IORCB for same task had abnormal end; x=internal flag for IOCAL; C=interrupt code applies to device other than one requested monopolizing control unit.

Figure 30. Fixed area of I/O request control block as set by IOCAL

Word 1		Word 2	
0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7	
Unchanged	Main storage address used for operation	Unchanged	

Figure 31. Channel command word list entry after task I/O interruption occurs

Example: Whenever you use an IOCAL, you should be sure to refer to the current version of the IORCB. The format of the IORCB is described by a dummy section in the system copy/macro library. You can get a copy by assembling a program with this instruction in it:

```
      COPY   CHAIOR
```

Suppose you want to read 120 bytes from symbolic device 85. You might write:

```
BGN       EX      0,TEST            EXECUTE IOCAL SVC
          B       AWAY
          DS      0D
TEST      IOCAL                     SVC 231 (IOCAL)
          DC      XL6'0'            NOT USED
          DC      AL1((TEST-IOREND)/64) LENGTH OF IORCB
*                                   IN 64-BYTE UNITS
          DC      XL2'0'            NO PAGE LIST
          DC      X'02'             PROTECTION KEY
          DC      X'0'              SIO FAILURE COUNT
          DC      X'01'             CCW LIST LENGTH (DOUBLEWORDS)
          DC      AL1((CCW-TEST)/8)  RELATIVE ORIGIN OF CCW LIST
*                                   IN DOUBLEWORDS
          DC      X'0'              RELATIVE ORIGIN OF STARTING
*                                   CCW IN LIST - FIRST ONE
          DC      AL1(120/8)        IORCB DATA BUFFER LENGTH IN DWORDS
          DC      AL1((BUF-TEST)/8)  RELATIVE ORIGIN OF IORCB
*                                   BUFFER IN DOUBLEWORDS
          DC      AL2(0)            NO ACTUAL ADDRESS
          DC      XL2'0'            NOT USED
          DC      AL2(85)           SYMBOLIC DEVICE ADDRESS
          DC      XL8'0'            NOT USED
          DC      V(POST)           V-CON OF POSTING ROUTINE
          DC      R(POST)           R-CON OF POSTING ROUTINE
          DC      XL28'0'           NOT USED
          DC      XL4'0'            NO OPTIONS
          DC      XL8'0'            NOT USED
ENDFIX    DS      0D                END OF IORCB FIXED AREA
BUF       DC      XL120'0'          BUFFER AREA
CCW       CCW     READ,0,X'20',120
IOREND    DS      0D                END OF IORCB
READ      EQU     194               2540 BCD READ COMMAND
```

Notice that the CCW page-list entry and displacement fields are both zero; this causes the IORCB buffer to be used and the information to be read into the first byte of the buffer. After the operation is complete, the supervisor causes a task-I/O interruption which stores the IORCB in the interruption storage area. The posting program (POST) can move the data out of IORCB buffer at that time.

ITI -- Inhibit Task Interruptions (O)

The ITI macro instruction is used to prevent the occurrence of all maskable task interruptions (external, asynchronous I/O, synchronous I/O, timer); it does this by setting the interruption storage area lock byte (ISALCK) to 1s with the Test and Set instruction.

Name	Operation	Operand
[symbol]	ITI	

Note: There are no operands.

To use ITI, you must define the symbol ISALCK by copying the interruption storage area dummy section (CHAISA) from the system macro/copy library.

LLIST -- Create Load List Entry (O)

LLIST generates an entry in a load list. A load list is used to determine during system startup which CSECTs and object modules will be loaded, and in what order, to form TSS. LLIST can be used when creating the load lists (object modules CHBVM, CHBRS, and CHBRC) that are furnished as part of the system on the IPL volume, or when creating load lists to be included in a delta data set. A load list in a delta data set will allow addition or deletion of system modules during a startup-to-shutdown session.

LLIST can also be used to insert user-accessible (usually PL/I) modules into IVM (initial virtual storage) so that they will be accessible for sharing by multiple users.

Name	Operation	Operand
csect or module name	LLIST	[load flag] [,selective load class] [,ROPROT=N]

Name	Operation	Operand
	LLIST	BARRIER={PRIVATE|SHARED}

csect or module name
: specifies a control section or object module name for which an entry is to be made in the load list. It may correspond to a control section already on a system load list (if this is a delta data set load list) or be a new control section to be inserted into TSS for this startup-to-shutdown session. When user-accessible modules are being listed, either the module name or the name of any CSECT may be specified; if any CSECT name is specified, the entire module is included during startup.

  Specified as: The name of the control section or object module.

load flag
: represents a load restriction or condition that must be observed when loading particular CSECTs.

  Specified as: One of the symbols designated below; they are shown with the hexadecimal code which is entered in the flag byte and the code meaning.

Symbol	Code	Meaning
SETXP	10	IVM pages; SETXP is allowed in CSECT.
USER	20	Module accessible to both user and system programmers.
RO	40	R/O control card (first module in resident supervisor that cannot be paged out by TSSS).
PGBD	80	Must be loaded on page boundary.

  Default: The load flag byte is set to zero, indicating no special restrictions.

selective load class
specifies a code that is used at startup to indicate selective loading. During startup, this code is compared to codes entered dynamically by the operator that indicate which modules are not to be loaded into the system. If the code in a load list entry matches an operator-specified code, the control section or module indicated in that entry is not loaded.

Specified as: A two-digit hexadecimal number. This may be specified as 00 or the operand may be omitted if the control section or module is always to be loaded, or any code from 40 to FF. Codes 01 to 3F are reserved for IBM system usage.

The following codes and system functions have been assigned:

LOAD LIST ID	FUNCTION	CONTROL SECTION(S)				
01	RJE	CEABAC	CEABBC	CEABCC	CHBRJE	
02	2250 SUPPORT	CZCVAC	CZCVAP	CZCVBC	CZCVBP	CZCVCC
		CZCVCP	CZCVDC	CZCVDP	CZCVEC	CZCVEP
		CZCVFC	CZCVFP	CZCVGC	CZCVGP	
08	ASSEMBLER	CEVMLL	CEVMLR	CEVPAR	CEVPAS	CEVPA1
		CEVPA2	CEVPA3	CEVPA4	CEVPA5	CEVPA6
		CEVPA7	CEVPA8	CEVPA9	CEVPB1	CEVPC1
		CEVPD1	CEVPE1	CEVPE2	CEVPF1	
09	FORTRAN	CEKADR	CEKAD2	CEKCSR	CEKJAR	CEKJB1
		CEKKRB	CEKKRR	CEKNG1	CEKNXR	CEKSAR
		CEKSF1	CEKTAR	CEKTCW	CEKUDR	CEKUKW
12	2305 SUPPORT	CEAB7C	CEAB8C	CEAB9C		
15	2860/2870/2880 ERROR RECOVERY	CMAH60C	CMAH70C	CMAH80C		
16	INTEGRATED CHANNEL ERROR RECOVERY	CMAHIC				
99	SIPE	CIPIOC				

Default: 00

ROPROT
indicates that the CSECT is not to have read only protection.

Specified as: N

Default: if not specified, page will be read only protected if CSECT has read only attribute.

BARRIER
generates a page that cannot be allocated so that if any reference is made to this page, program interrupt 5 (addressing) will occur.

Specified as:

PRIVATE - for private pages
SHARED - for shared pages

Default: none.

CAUTIONS: When changing the load list, the programmer should be careful to place entries into the modification CSECT in the proper sequence (some load-list entries must appear in the CSECT before other entries). For example, the RTAM buffer pages and control blocks must be loaded before other RTAM modules.

LLIST can not be used for control section or module names that contain characters not acceptable to the TSS assembler (for example, %, _, *). The load list entries for these CSECT or module names must be coded using DC instructions. For example:

```
DC      CL8'%SECTNM'    MODULE/CSECT NAME
DC      XL1'00'         RESERVED
DC      XL1'20'         LOAD FLAG
DC      XL9'00'         RESERVED
DC      XL1'40'         SELECTIVE LOAD CLASS
DC      YL1'00'         EXTENDED LOAD FLAG
DC      3XL1            SPARE
```

Programming Notes: There are three system VPAM data sets that contain load lists: TSS*****.SYSIVM, TSS*****.RESSUP, and TSS*****.RSSSUP. SYSIVM contains all of the CSECTs that are part of initial virtual storage (IVM); RESSUP all the CSECTs that are part of the resident supervisor; and RSSSUP all the CSECTs that are part of the RSS support. Each of these data sets has a special member (the load list) that contains the names of the CSECTs for the members within the partitioned data set that are to be loaded into the system at startup.

System programmers can modify these load lists by creating a new load-list CSECT (see below) and including it as a delta data set on the delta data set volume that is to be processed during startup (see System Generation and Maintenance). Delta data sets are used to dynamically modify the system to determine the effect of changes being introduced to the system without permanently updating the system. These changes remain in effect from startup to shutdown.

During startup, the delta data set volume is searched for IVM, RESSUP, and RSS control sections. The IPL volume is also searched for any remaining system CSECTs for which no delta data set exists. Any CSECT found on the delta data set volume having the same name as a CSECT that is part of one of these system data sets replaces the existing CSECT member of that data set; it is loaded with the control sections found on the IPL volume to make up the current version of the system data set.

Since a new load list actually replaces the old one, all the old load-list entries that the user wants to have included in the new load list must be respecified in the new load-list CSECT. To make sure all old entries are specified, a programmer should create the new CSECT by copying the old load list and editing it to include any new entries or delete specific old entries. The source data sets that the programmer might want to copy and edit are: SOURCE.CHBVM for the SYSIVM data set, SOURCE.CHBRC for the RESSUP data set, and SOURCE.CHBRS for the RSSSUP data set.

When creating a new load-list module, assign the same CSECT and module name as that of the load-list module that is to be replaced.

The LLIST macro instruction can be used to generate entries in a load-list CSECT being created by system programmers. Each 24-byte load-list entry contains an 8-byte CSECT name, a reserved byte, a startup flag byte, nine bytes that are not used until startup, and a selective loading byte, as shown in Figure 32.

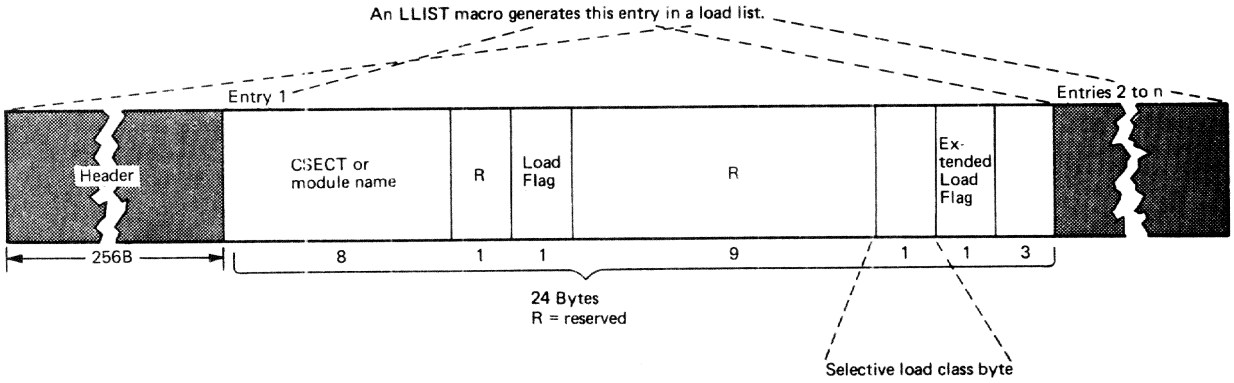


Figure 32. Load list entry

Example: A system programmer wants to load two new modules into initial virtual storage (IVS) during startup. He modifies SYSIVM.LOADLIST using the LLIST macro.

```
*** CSECT FOR SYSIVM.LOADLIST DELTA DATA SET **********************

CHBVML      CSECT
            .             ALL ENTRIES FROM OLD LOAD LIST
            .             THAT ARE TO BE INCLUDED IN
            .             THE NEW ONE
            .
CHBNEW1     LLIST   PGBD,40
```

The LLIST generates:

```
CHBNEW1   DC      CL8'CHBNEW1'    CSECT NAME IN NEW MODULE
          DC      XL1'0'          RESERVED
          DC      XL1'80'         LOAD FLAG
          DC      XL9'0'          RESERVED
          DC      XL1'40'         SELECTIVE LOAD CLASS
```

LOCPAG -- Locate Page (R)

The LOCPAG macro sets up the parameters and calls the Locate Page Module (CEAML) to obtain the addresses of the Page Table (PGT) and External Page Table (XPT) for a specified virtual memory address and task. Optionally, the page hold count for the specified page is either incremented or decremented.

Name	Operation	Operand
[symbol]	LOCPAG	virtual memory address, TSI address [,error address] [,READ={Y|N}] [,HOLD={IGNORE|INCR|DECR}]

virtual memory address
specifies the virtual memory address to be located

Specified as: A register containing the virtual memory address or a symbol, defined as fullword aligned, containing the virtual memory address.

TSI address
specifies the real memory address of the TSI of the owner of the virtual memory address.

Specified as: A register containing the TSI address or a symbol, defined as fullword aligned, containing the TSI address.

error address
specifies where to branch to if an error indication was returned by Locate Page (CEAML).

Specified as: A symbolic address.

Default: The program continues at the next sequential instruction.

READ=
indicates whether or not the page table describing this virtual memory address is to be read into main storage if it is not currently in main storage.

Specified as: Y or N.

Default: Y

HOLD=
indicates whether the page hold count for this virtual memory address should be ignored, incremented, or decremented.

Specified as: IGNORE, INCR, or DECR

Default: IGNORE.

Execution: Upon execution of this macro instruction, parameter registers are set up and the Locate Page Module (CEAML) is called.

Return Data:

Register	Contents	Meaning
0	page table address	---
1	external page table address	---
15	0	no error
	4	segment not assigned
	8	page not assigned
	12	page table not in memory

LOGVLOCK -- Define VM Lock Anchor (O)

LOGVLOCK defines a block of storage where other xxxVLOCK macros can record the status of a VM Lock.

Name	Operation	Operand
symbol	LOGVLOCK	

Note: The name symbol is required; there are no operands.

Execution: This macro generates a control block for use by other macros of the xxxVLOCK set.

CAUTION: This macro must appear in a PSECT.

Programming Note: Refer to VM Locking in Section 3.

LVPSW -- Load Virtual Program Status Word (R)

The LVPSW macro instruction alters the flow of your program by changing its VPSW.

Name	Operation	Operand
[symbol]	LVPSW	[VPSW address]

VPSW address
specifies the virtual storage address at which the new VPSW is presently stored.

Specified as: An RX address, or as register notation.

Default: It will be assumed that the issuer has placed the address of the VPSW in register 1.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding LVPSW. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding LVPSW must be issued with the PRIVILEGED option.

Execution: The virtual program status word whose address is specified becomes the current virtual program status word. The previous contents of the virtual program status word are lost.

Example: Assume location NEWVPSW contains a new virtual program status word that is to be loaded. You might write:

```
        NAME    LVPSW    NEWVPSW
```

MAPTDY -- Connect, Disconnect, or Expand the TDY (O)

The MAPTDY macro instruction transfers control to the maptdy program in Initial Virtual Storage.

Name	Operation	Operand
[symbol]	MAPTDY	{T|U|E}

T|U|E
specifies the function to be performed by the maptdy program.

Specified as:

T - disconnect user space and connect TDY.

U - disconnect TDY and connect user space.

E - expand TDY area.

Execution: This macro instruction places in register 1 the following values for the requested function:

request	value in register 1
T	1
U	2
E	3

Control is then passed via type 1 linkage to module CZCDM. Upon return from CZCDM, the requested function will have been performed.

There is no return data.

Programming Notes: if the TDY is already connected on a 'T' call or the user space is already connected on a 'U' call, CZCDM just returns to the caller. If the expand TDY request 'E' is specified, the TDY must be connected prior to the 'E' call.
The disconnected segment group name for the TDY area is defined in the ISA by the field ISATDYN. The flag ISATDCM defined in the ISA is used to determine if the TDY or the user space is connected; if the flag is on, the TDY is connected.

MOVGQE -- Move GQE to New Scan Table Entry

MOVGQE sets up the parameter list and calls CEAJMG to move the GPE to a new scan table entry, or if finished, to release the GQE and certain attached control blocks.

Name	Operation	Operand
[symbol]	MOVGQE	GQE

GQE
generalized queue entry to be moved.

Specified as: the address of the GQE, or the name of a symbol defined as a fullword containing the address of the GQE.

CAUTIONS: The PSA dsect CHAPSA must have been copied and assigned a base register prior to the use of this macro. The expansion of this macro is affected by the use of the CVT macro prior to the use of this macro. MOVGQE assumes that the GQE being moved is not currently enqueued on the scan table.

Programming Notes: MOVGQE will move the GQE loc-on-queue values to get the next queue entry. If GQELQ is not equal to X'FFXX', the GQE will be enqueued on the corresponding entry. If GQELQ is equal to X'FFXX', MOVGQE will release the GQE and any associated PCBs. If there are no PCBs, then the caller must make sure that GQECNT is 0.

Examples:

(1) Register notation; CVT YES is assumed

```
    MOVGQE (1)
       .
       .
       .
    *  L  15,CVTJMG
    *  BALR 14,15
```

(2) Symbolic name; CVT YES is assumed

```
    MOVGQE TCWGQE    where TCWGQE is defined as a fullword
       .
       .
       .
  + CHDVAL TCWGQE,1
  + L  1,TCWGQE
  + L  15,CVTJMG
  + BALR 14,15
```

(3) Symbolic name; CVT NO is assumed

```
    MOVGQE TCWGQE
       .
       .
       .
  + CHDVAL TCWGQE
  + L  1,TCWGQE
  + L  15,PSACVT
  + L  15,CVTJMG-CHACVT(,15)
  + BALR  14,15
```

MOVXP -- Move Page Table Entries (R)

The MOVXP macro instruction moves page table and external page table entries from one table to another or from one part of a table to another.

Name	Operation	Operand
[symbol]	MOVXP	[old address][,new address][,page count]

old address
specifies the virtual storage address associated with the first page table entry or external page table entry you want moved.

Specified as: An RX address, or register notation. The address must be a multiple of 4096.

Default: It will be assumed that the issuer has placed the new address in register 0.

new address
specifies the new virtual storage address with which you want the first entry associated.

Specified as: An RX address, or register notation. The address must be a multiple of 4096.

Default: It will be assumed that the issuer has placed the old address in register 1.

page count
specifies the number of consecutive entries you want moved.

Specified as: An absolute expression or register notation.

Default: It will be assumed that the issuer has placed the page count in register 15.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding MOVXP. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding MOVXP must be issued with the PRIVILEGED option.

Execution: The page table and external page table entries associated with the page address specified in the first operand are now associated with the page address specified in the second operand. The number of entries to be moved is provided in the third operand. Each old page table entry is marked assigned but unavailable; each old external page table entry is cleared to zero.

Example: Suppose you want to move 300 pages located at IN to an area beginning at OUT; both IN and OUT must be page boundary addresses. You might write:

```
MOVE    MOVXP    IN,OUT,300
```

MSGWR — Issue System Message and Get Response (S)

Note: This macro has been replaced by PRMPT and must not be used for new code. This documentation is retained only to aid in the maintenance of existing programs.

The MSGWR macro instruction issues a message to SYSOUT and, if specified, fetches the response and places it in a user-designated area. In conversational mode, SYSOUT is considered to be the terminal and, in nonconversational mode, SYSOUT is considered to be the data set containing system messages that will be printed at the end of the task. Responses can only be made in conversational mode; therefore, the response option may only be specified in conversational mode. If the response option is specified in nonconversational mode, the task is abnormally terminated.

MSGWR issues system messages only. The text and the message numbers assigned to these messages are described in System Messages. The user can modify a standard system message by inserting variable information into it. Variable fields are filled in by MSGWR, using information supplied by the user.

The format for the MSGWR macro instruction is indicated below. The information associated with each parameter must be placed in storage by the programmer before issuing the macro instruction.

Standard form:

```
|Name     |Operation|Operand                                                  |
|---------|---------|---------------------------------------------------------|
|[symbol] |MSGWR    |message id,[variable data]                               |
|         |         |[,response area,response length]                         |
```

L- and E-form:

```
|Name     |Operation|Operand                                                  |
|---------|---------|---------------------------------------------------------|
|[symbol] |MSGWR    |[message id],[variable data]                             |
|         |         |[,response area,response length],MF={L|(E,list)}         |
```

Note: A symbol is required in the name field of the L-form. If the message id operand is not specified in the L-form, it must be supplied in the E-form.

message id
specifies the address of a fullword containing the message number, a response flag, and the number of variable fields to be inserted. This information must be inserted by the user in the fullword in the format described below.

Byte	Contents
0-1	Message number - four hexadecimal digits
2	Response flag - 1 if response is desired 0 if not
3	Numbers of variable fields to be inserted (see the description of the variable data operand)

Specified as: In the standard and L-form, as a relocatable expression; in the E-form only, also as an RX address.

variable data
specifies the address of one or more doublewords that identify the text to be inserted in the variable field of the message. There must be as many doubleword entries as are specified in byte 3 of the location specified by the message id operand. The first of these words gives the number of bytes of text, in binary. The second word points to the actual text.

Specified as: Same as the first operand.

response area
specifies the address of the area into which the response (if any) is to be placed.

Specified as: Same as the first operand.

response length
is the address of a one-word field into which the length of the response (if any) is to be placed. The response area must be large enough to accept the longest expected reply (128 bytes). If the response does not fit in the allotted area (because less than 128 bytes were specified), it will be truncated, starting with the rightmost character.

If the user has any doubt about the length of the response, he should use a 128-byte response area.

Specified as: Same as the first operand.

Initialization: If this macro instruction is to be executed in a privileged module, the most recently issued DCLASS macro instruction in the assembly must have specified PRIVILEGED. Also, the address of a save area must be placed in register 13 before this macro instruction is executed.

Programming Notes: Registers 2 through 12 and the floating-point registers are unaffected by expansion of the MSGWR macro instruction.

Return Data: On return from MSGWR, a hexadecimal code is loaded into the low-order byte of register 15. The significance of these codes is as follows:

Code	Significance
00	No attention interruption; no error in response length (if applicable).
04	Response too long for area specified. Truncation occurred.
08	Attention interruption occurred; status of response (if any) is unpredictable.

Upon return from MSGWR, if a response was requested, the actual byte length of the response is placed in the field specified by the response length operand.

Example: In the following example, the system message D001 is to be written on the terminal with a variable field containing XXUSERID; the expected response should contain a maximum of 8 characters, and the response is to be placed in an area called READIN.

```
EX1     MSGWR MSGCD,VARFLD,READIN,RLENGTH
```

In this example, the user has provided information required by the MSGWR macro instruction elsewhere in his program through use of DC innstructions. The parameters of the MSGWR macro instruction have been specified using the symbolic addresses of the DC instructions.

```
MSGCD       DC    X'D0010101'
VARFLD      DC    F'8'
            DC    A(TEXTVA)
TEXTVA      DC    C'XXUSERID'
```

```
READIN      DC    2F'0'
RLENGTH     DC    F'8'
```

NIB -- Generate Node Identification Block (S)

This macro instruction is completely documented in the Assembler User Macro Instruction manual, except for several operands that are available only to the systems programmer. Only the definitions and specifications for these additional operands are given below, but for continuity, all the operands are shown in the metalanguage that follows.

```
┌──────┬─────────┬──────────────────────────────────────────────────────┐
│Name  │Operation│Operand                                               │
├──────┼─────────┼──────────────────────────────────────────────────────┤
│symbol│NIB      │[RNAME=resource name]                                 │
│      │         │[,ADDR=({SDA=sda|RID=network address})]               │
│      │         │[,USN=user number] [,CP=component number]             │
│      │         │[,OPTION=(PASS,IN,OUT,DFT,REPL,DSN,AQ)]              │
│      │         │[,EXLST=address of exit list]                         │
│      │         │[,LOGON=address of logon parameter list]              │
│      │         │[,LASTED={Y|N}] [,FCL=address of FCL name]            │
│      │         │[,RTE=address of routing table]                       │
│      │         │[,MF={L|(E,address of NIB)}]                          │
└──────┴─────────┴──────────────────────────────────────────────────────┘
```

ADDR
: specifies the address and address type assigned to the node.

SDA
: specifies the symbolic address by which TSS knows the node.

  Specified as: the address of a character string of 2 to 4 hexadecimal characters preceded by one byte containing the length of the string.

  Default: none.

RID
: specifies the network address by which TSS knows the node.

  Specified as: the address of a character string of 1 to 17 alphanumeric characters in the form 'major node.minor node', preceded by a one byte field containing the length of the character string.

  Default: none.

FCL
: specifies the name of the FCL to be used to control the connected node. The OPNDST macro processor will issue a explicit LOAD macro instruction to load the FCL and its accompanying Format Control Module.

  Specified as: the address of a 1 to 8 byte name preceded by one byte containing its length; or register notation (2 through 12); or an RX address.

  Default: none.

RTE
: specifies the address of a routing table to be used by TAMII for connecting to the node. The routing table is described in the CHARTE DSECT. The RTE operand is valid only for OPTION=AQ, and will be ignored for any other OPTION.

Specified as: register notation (2 through 12); or an RX address.

Default: none.

OCBD -- Specify OS DCB DSECT

The OCBD macro instruction enables the issuer to gain symbolic access to the fields in the OS DCB.

Name	Operation	Operand
	OCBD	

Note: There are no operands.

Execution: The OCBD macro expands into a copy of the OS DCB dsect.

Programming Notes: It is used in the Program Product Language Interface (PPLI) to analyze symbolically the OS description of the userS request. It should not be used in assembling or executing OS programs under PPLI since it is a TSS dsect; the OS DCB/DCBD macro should be used instead. These OS macros reside in the SYSOS.MAC library which is referenced when using the ASM H program product.

OPEN (MSAM) -- Prepare the Data Control Block for Processing (S)

The OPEN macro instruction initializes one or more data control blocks for processing of their associated data sets. This description is for system programmers processing MSAM data sets.

Standard form:

Name	Operation	Operand
[symbol]	OPEN	(dcb address,[({INPUT|OUTPUT})],...)

L-form:

Name	Operation	Operand
symbol	OPEN	[(dcb address,[({INPUT|OUTPUT})],...)]MF=L

Note: A symbol is required in the name field. Any operands omitted must be specified in the E-form.

E-form:

Name	Operation	Operand
[symbol]	OPEN	[(dcb address,[({INPUT|OUTPUT})],...)MF=(E,list)

Note: If E-form operands are specified, they will overlay corresponding operands specified in the L-form. The list operand must specify the symbol in the name field of the L-form; or the symbol may be loaded into register 1 and the list operand specified as (1).

dcb address
specifies the address of the data control block to be initialized.

Specified as: In the standard and L-form, a relocatable expression; in the standard and E-form, register notation (2 through 12); in the E-form only, also as an RX address. If register notation is used, the address must first be loaded into the specified register.

INPUT|OUTPUT
specifies whether the associated data set is for input or output.

Specified as: INPUT or OUTPUT (see the programming notes).

Default: INPUT

CAUTION: The following errors cause the results indicated:

Error	Result
Opening data control block that is already open.	No action.
Specifying address of invalid data control block.	Task terminated.
Opening data control block when DDNAME has not been provided.	Task terminated.
Opening data control block when corresponding DDEF macro instruction or command has not been provided.	Task terminated (prompting will be given if task is conversational).
Opening data control block containing invalid DSORG specification.	Task terminated.

Programming Notes: You may specify any number of data control block addresses and associated options in the OPEN macro instruction. This facility allows parallel opening of data sets.

OPEN initializes all the fields in the MSAM portion of the DCB, as well as obtains all the pages necessary for MSAM operations.

If the DCB COMBINE flag is set, the reader is assumed to be on the same 2540 frame as the punch, and the symbolic device address of the reader must be one greater than that of the punch.

A violation of any of the following restrictions causes the OPEN macro instruction to abnormally terminate the task.

- The DCB MACRF field must specify only that GET or PUT macro instructions will be issued.
- The DCB DEVD field must specify (possibly from the DDEF command) a card reader, card punch, or printer, and this device must correspond to the device specified in the DDEF command.
- If the device is a card reader, the data set must be opened for input.
- If the device is a card punch or printer, the data set must be opened for output.
- The DCB RECFM field must indicate fixed-format records or variable-format records; A/M control characters may also be specified.
- The DCB DEVD must specify the card punch if the DCB COMBINE flag is set.

OPENLOCK -- Reset a Resident Supervisor Lock Byte (O)

The OPENLOCK macro instruction resets to X'00' a byte set on as the result of a SETLOCK macro instruction (see the SETLOCK macro instruction).

Name	Operation	Operand
[symbol]	OPENLOCK	lockbyte,module ID,error number[,{LOG|NOLOG}]

lockbyte
specifies the byte that is to be reset.

Specified as: A symbolic address.

module ID
is included for compatibility with the macro definition and has no function.

Specified as: A two-digit decimal number.

error number
is included for compatibility with the macro definition and has no function.

Specified as: A two-digit decimal number.

LOG|NOLOG
specifies whether the address of the OPENLOCK macro instruction is to be entered in the logging field associated with the lock byte.

Specified as: LOG or NOLOG

Default: NOLOG

OPNVLOCK -- Open VM Lock (O)

OPNVLOCK is used to open a VM Lock previously set by SETVLOCK.

Name	Operation	Operand
[symbol]	OPNVLOCK	log [,OPEN=open]

log
specifies the VM Lock to be opened.

Specified as: the symbol naming a LOGVLOCK macro.

open
specifies an address in the current module to be branched to if the specified lock is already marked "open".

Specified as: an RX address.

Default: The status of the lock will not be cheched.

Execution: If the branch address is specified and if the VM Lock Anchor indicates "open", the branch will be performed. Otherwise, the indi-

cated VM Lock will be opened and the VM Lock Count (ISAVLKCT) in the task's Interrupt Storage Area (CHAISA) will be decremented.

CAUTION: This macro must be protected from task interrupts by ITI/PTI.

Programming Note: Refer to VM Locking in Section 3.

PCSVC -- Enter Program Control System (0)

The PCSVC macro instruction assembles a constant hexadecimal machine instruction that is intended to be implanted in a user's nonprivileged code.

Name	Operation	Operand
[symbol]	PCSVC	

Note: There are no operands.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a module prior to coding PCSVC. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding PCSVC must be issued with the PRIVILEGED option.

CAUTION: The SVC 125 generated by PCSVC can only be executed in non-privileged code.

Execution: A task-SVC interruption is created to transfer control to the task monitor. The task monitor then transfers control to the program control system (PCS). This SVC is used by PCS to replace user instructions in response to the AT command (see Command System User's Guide).

Example: Suppose you want to plant a transfer of control; you might code:

```
MOVE    MVC     NAME(2),PLANT
        B       AWAY
PLANT   PCSVC
```

Programming Notes: Although the PCSVC macro instruction must be assembled in a CSECT in which a DCLASS PRIVILEGED macro instruction has been previously issued, PCSVC cannot be executed in a privileged CSECT. It must be executed in nonprivileged code; it is usually implanted in nonprivileged code by first assembling it in privileged code and then moving its assembled hexadecimal machine instruction into the nonprivileged code.

PGOUT -- Write Virtual Storage Pages to External Storage

The PGOUT macro instruction enables you to write from one to eight virtual storage pages to one or more external storage devices.

Name	Operation	Operand
[symbol]	PGOUT	

Note: There are no operands.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a module prior to coding PGOUT. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding PGOUT must be issued with the PRIVILEGED option.

The PGOUT macro instruction must be the subject of an Execute instruction and must occupy the high-order halfword of the first word of a parameter list called an I/O paging control block (IOPCB). The IOPCB consists of a header and a number of external storage list entries (see Figure 33).

Execution: The resident supervisor reads into storage any pages in the list that aren't already in main storage; when all pages are in, the supervisor writes them out at the external storage locations supplied in the external storage list. From one to eight consecutive virtual storage pages may be transmitted; the destination external storage locations need not be consecutive and may be on different devices.

Return data: Before returning, the resident supervisor puts information in register 0 to describe the action with each page in the external storage list. Four bits of register 0 are assigned to each page; bits 0-3 for the first page, bits 4-7 for the second, etc. The four bits are interpreted as follows:

Value	Meaning
0000	No error - page transmitted
0011	Virtual storage page not assigned to task
0100	Request for zero pages
0101	Symbolic device not assigned to task
0110	Page in -- bad device -- volume is movable
0111	Page in -- bad device -- volume is fixed
1000	Page in -- medium failure
1001	Page out -- bad device -- volume is movable
1010	Page out -- bad device -- volume is fixed
1011	Page out -- medium failure

Example: Suppose you want to write virtual storage page RSLTS on the 127th page position of symbolic device 34. You might write:

```
OUT     EX      0,MOVE
        B       SOMEPLACE
MOVE    PGOUT
        DC      H'1'          1 PAGE TO BE TRANSMITTED
        DC      A(RSLTS)      VIRTUAL MEMORY ADDRESS
        DC      H'34'         SYMBOLIC DEVICE NUMBER
        DC      H'127'        RELATIVE PAGE NUMBER
```

Format of I/O Paging Control Block Header	
HalfWord 1	HalfWord 2
PGOUT -- SVC 242	Number of ESL entries
Virtual storage address of first of 1-8 pages to be transmitted	

Format of External Storage List Entry (Maximum of Eight)	
Two Bytes	Two Bytes
System symbolic device number	Relative page number

Figure 33. I/O paging control block

PR -- Print a Data Set (S)

This macro instruction is completely documented in the Assembler User Macro Instruction manual, except for one operand that is available only to a systems programmer. The definition and specification for only that one operand are given below, but for continuity, the metalanguage format that follows shows all the operands.

Standard form (see 'operand strings' in Part 2, Section 1.)

Name	Operation	Operand
[symbol]	PR	{address of operand string |'DSNAME=data set name [,STARTNO=starting position] [,ENDNO=ending position] {,PRTSP={EDIT|1|2|3} |[,HEADER=H] [,LINES=lines per page] [,PAGE=P]} [,ERASE={Y|N}] [,ERROPT={ACCEPT|SKIP|END}] [,FORM=standard default region name] [,STATION=station id] [,TAPOPT={AC|AD|AE|ED|EC}] [,CHARS=([GS1] [,GS2] [,GS3] [,GS4])] [,FCB=fcb name] [,PAPER=paper type] [,COPIES=(nnn[,(GP,...)])] [,FLASH=(overlay name[,COUNT])] [,SYSUCS=users sysucs dsname] [,BURST={Y|N}] [,COPYMOD=copy modification data set name] [,TRC={Y|N}]} [,NPRIORTY=transmission priority] [,NETACCT=network account number] [,DELIVER=([prgmrnam][,room][,dept][,bldg])] [,PRTCLASS=printing output class] [,INDEX=indexing offset] [,EXTWTR=external writer name] [,MODTRC=table reference character]'}

L-form (see "Operand Strings" in Part II, Section 1):

Name	Operation	Operand
symbol	PR	{address of operand string |'DSNAME=data set name [,STARTNO=starting position] [,ENDNO=ending position] {,PRTSP={EDIT|1|2|3} |[,HEADER=H] [,LINES=lines per page] [,PAGE=P]} [,ERASE={Y|N}] [,ERROPT={ACCEPT|SKIP|END}] [,FORM=standard default region name] [,STATION=station id] [,TAPOPT={AC|AD|AE|ED|EC}] [,CHARS=([GS1] [,GS2] [,GS3] [,GS4])] [,FCB=fcb name] [,PAPER=paper type] [,COPIES=(nnn[,(GP,...)])] [,FLASH=(overlay name[,COUNT])] [,SYSUCS=users sysucs dsname] [,BURST={Y|N}] [,COPYMOD=copy modification data set name] [,TRC={Y|N}]} [,NPRIORTY=transmission priority] [,NETACCT=network account number] [,DELIVER=([prgmrnam][,room][,dept][,bldg])] [,PRTCLASS=printing output class] [,INDEX=indexing offset] [,EXTWTR=external writer name] [,MODTRC=table reference character]'} ,MF=L

Note: A symbol is required in the name field of the L-form.

E-form

Name	Operation	Operand
[symbol]	PR	MF=(E,list)

TAPOPT

(See 'Extended PRINT Command Facilities' in Part 1, Section 4 for a description and specification of the TAPOPT operand.)

PRESENT -- Present Current Schedule Level (R)

The PRESENT macro instruction enables a task to find out its current schedule table entry (STE) level.

Name	Operation	Operand
[symbol]	PRESENT	

Note: There are no operands.

Execution: The PRESENT macro instruction generates a CHANGE SVC (SVC 227) specifying a schedule level greater than the schedule table limit. The SVC executed by CHANGE causes an interruption. When the resident supervisor receives the interruption, it examines the specified change level to determine if it is greater than 255. If it is, no change to the current level is made; instead, the current schedule level for the task is returned in register 15.

Return Data: Bits 8-15 of register 15 contain the current schedule table level. All other bits should be ignored.

Example: To determine the current schedule level of a task, write:

```
NAME    PRESENT
```

PTI -- Permit Task Interruptions (0)

The PTI macro instruction cancels the effect of an ITI macro instruction; it allows pending task interruptions to occur (if the task-mask bits in the VPSW are 1s).

Name	Operation	Operand
[symbol]	PTI	

Note: There are no operands.

PULSE -- Pulse Schedule Table Entry Level (0)

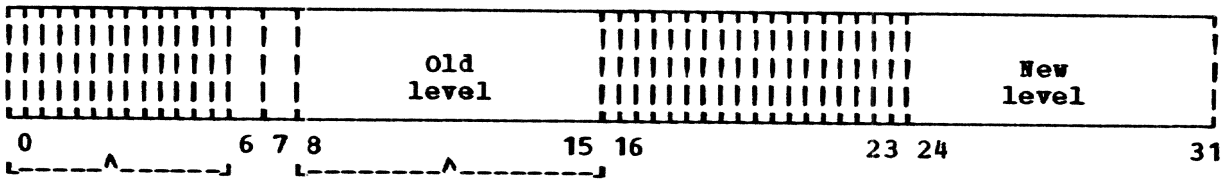
The PULSE macro instruction changes a task's schedule level to a preset pulse level that is associated with its current level entry.

Name	Operation	Operand
[symbol]	PULSE	

Note: There are no operands.

Execution: When the resident supervisor receives the SVC interruption generated by this macro instruction, it changes the task's schedule level to that specified in the current schedule level's pulse field. If the pulse field contains a schedule level that is invalid (that is, outside the schedule table limits, or zero), it is replaced with the current level, and a minor SYSER is issued. Register 15 contains an error indicator upon return from SVC 226.

Return Data: The format of the return information in register 15 appears as follows:



Validity byte value:

Setting	Meaning
All zeros	The new pulse level is valid.
Bit 6 (on)	The pulse level was outside the limits of the schedule table.
Bit 7 (on)	Zero pulse level was specified; no change of levels occurred.

Example: Suppose your task is about to perform a function that requires longer though less frequent time slices, and that the pulse field associated with your current schedule level has been set to accommodate your special need. You may issue the following macro instruction to change to the desired pulse level:

```
BUD     PULSE
```

PURGE -- Purge I/O Operations (R)

The PURGE macro instruction suppresses I/O devices and/or removes them from your task's symbolic device list.

Name	Operation	Operand
[symbol]	PURGE	(action[,device number]),(task[,taskid])

action
specifies the purging action you want.

Specified as: One of the codes defined below. If SL, SD, SR, or SS is specified, the device number must be specified.

AR - Purge all devices immediately, removing the TSDL entries.

AL - Purge all devices immediately, leaving the TSDL alone.

AD - For all devices, remove the TSDL only.

AS - Purge all devices, letting active devices quiesce.

SL - Purge single device immediately, leaving TSDL alone.

SD - Remove TSDL for single device.

SR - Purge device specified in parameter 2 immediately, removing the TSDL entry.

SS - Purge device specified in parameter 2, letting the device quiesce if active.

Default: It is assumed that register 0 has been loaded with the action code in bytes 0 and 1, and the symbolic device address in bytes 2 and 3. If the pre-loaded action code is 'Ax', the SDA must be 0.

device number
specifies the symbolic device address of the device to be purged. The device number must be specified for action codes SL, SD, SR, or SS.

Specified as: a number or absolute expression specifying the register (2 through 12) which contains the SDA.

Default: see "action".

task
specifies whether the purge is for all tasks or a particular task.

Specified as:

AT - for all tasks

ST - only for a particular task

If ST is specified, the taskid must be specified.

Default: It is assumed that register 1 has been joaded with the task code in bytes 0 and 1, and the taskid in bytes 2 and 3. If the pre-loaded action code is 'Ax', the TASKID must be 0.

taskid
specifies a task ID for which the purge is to be effective.

Specified as: a number or absolute expression specifying the register (2 through 12) which contains the TASKID.

Default: see "task".

Execution: The I/O devices to be purged are suppressed and/or removed from the task symbolic device list (TSDL) of the task or tasks to which the purge is to apply. If a device is to be allowed to quiesce, its task symbolic device list entry is merely suppressed; if a device is to be purged immediately, its task symbolic device list entry is removed from the task symbolic device list. The TSDLs to be used depend on whether one or all tasks are to have their I/O devices purged.

Return Data: Register 0 contains an error flag, if applicable. Depending on the request, this bit can have these interpretations:

For one device, one task -- device not assigned to task.

For all devices, one task -- no task symbolic device list exists.

For all tasks, one or all devices -- devices not assigned to any task.

If a task that does not have the system operator privilege issues PURGE for all devices and all tasks, a system error code of 6101 is generated.

Example: Suppose you wish to purge the I/O device assigned symbolic device number 357 for any tasks that might be using it, but you are willing to wait for the device to quiesce. You might write:

```
        LH    2,=Y(357)
PUR     PURGE SS,(2),AT
```

PUT (MSAM) -- Put a Record (R)

The PUT macro instruction may be specified in either the locate mode or the move mode. When you specify the macro instruction in the locate mode, PUT returns, in register 1, the address within an output buffer of an area large enough to contain an output record; you should then construct, at that address, the next sequential logical record of the output data set. When you specify the macro instruction in the move mode, PUT moves the next sequential logical record of the output data set from the location you have specified into an output buffer.

Name	Operation	Operand
[symbol]	PUT	dcb address[,area]

dcb address
: specifies the address of the data control block opened for the data set being processed.

  Specified as: An RX address, or register notation (1 through 12). Execution time is saved if register 1 is specified. If register notation is used, the address must first be loaded into the specified register.

area
: specifies the address of the next logical record to be moved into the output buffer. This operand is used only when the macro instruction is specified in the move mode.

  Specified as: An RX address, or register notation (0 or 2 through 12). Execution time is saved if register 0 is specified. If register notation is used, the address must first be loaded into the specified register.

CAUTION: If any field of the DCB is altered by an improper source, the task may be abnormally terminated when a PUT macro instruction is executed.

Execution: Upon completion of the PUT macro instruction, a code indicating the manner in which the instruction was completed is returned in register 15. The codes and their meanings are given in Figure 34.

Return Code	Meaning
0	Operation completed successfully.
4	I/O not complete; the record has not been accepted, since there is no room remaining in the present buffer and the next sequential buffer has not yet been released from the previous I/O request. The PUT macro instruction should be reissued. (See the discussion of "Interruption Entry Handling.")
8	Unrecoverable I/O error occurred, and the record was not accepted. Register 1 points to the record on which the I/O error occurred -- in the case of an equipment check on the card punch, register 1 points to the record immediately following that on which the error occurred -- and register 0 points to associated DECB. FINISH and/or CLOSE macro instructions may be issued. However, if the I/O error is not permanent (DEBNF2 or DECG1 not on), you may continue processing records beyond the one that failed by reissuing the PUT.

Figure 34. Return codes for MSAM PUT macro instruction

Programming Notes: The length of the logical record is determined by the value of the LRECL field of the data control block for fixed-length (format-F) records and by the value of the control bytes for variable-length (format-V) records. If you write format-V records, the value of the LRECL field must be set equal to the maximum length of the logical record prior to the locate-mode PUT macro instruction. The value may be changed between executions of the PUT macro instruction and will be used to determine when to truncate the present buffer. The control program uses the current value of the LRECL field to determine the amount of buffer space needed for the record, even though the actual length is determined by the control byte built by the user in the buffer area after the completion of the locate-mode PUT macro instruction.

If you do not specify FORTRAN (ASA) or machine code in the RECFM field of the data control block, the PRTSP and STACK fields of the DCB are used to control line spacing and stacker selection, respectively.

Printer: The MODE field of the data control block is not referred to for the printer. The LRECL field of the data control block must be set to a value not exceeding 133 bytes for format-F records, and 137 bytes for format-V records. These values are 132 bytes and 136 bytes, respectively, if the FORTRAN or machine code was not specified in the RECFM field of the data control block. If you use control characters (A or M) for carriage control, channel 12 is ignored (greater efficiency is achieved by not having a channel 12 punched on the carriage control tape). However, if you do not use control characters and channel 12 is sensed, an immediate eject to channel 1 is performed.

Card Punch: For format-F records, you must set the LRECL field of the data control block to a value not exceeding 81 bytes for EBCDIC, and 161 bytes for column binary. These values are 80 bytes and 160 bytes, respectively, if the FORTRAN or machine code was not specified in the RECFM field of the data control block. For format-V records, you must set (for locate mode only) the LRECL field to a value not exceeding 85 bytes for EBCDIC or 165 bytes for column binary; these values are 84 and 164 bytes, respectively, if the FORTRAN or machine code was not specified in the RECFM field. The MODE field of the data control block must contain a binary 0 for EBCDIC or a binary 1 for column binary.

Example: In the following example, which illustrates the use of both the locate-mode and move-mode PUT macro instructions, you want to print a file of 132-byte EBCDIC records. After each line is printed, one line is spaced. Since the return codes provided by the macro instruction are multiples of 4, it is possible for you to set up a branch table to provide proper control of processing.

```
JHL         DCB   DSORG=MS,MACRF=P,           BUILD DCB
                  DDNAME=TODD,DEVD=PR,
                  PRTSP=1,RECFM=F,LRECL=132
            OPEN  (JHL,(OUTPUT))              OPEN DCB
            .
            .
            .
            .
            LA    3,RCTABLE                   SET UP BRANCH TABLE
            LA    1,JHL                       LOAD ADDR OF DCB
MOVE        PUT   (1),WORK                    MOVE-MODE PUT MACRO
            L     5,0(15,3)                   BRANCH ON RC INDEX AND
            BR    5                           RCTABLE AS BASE
            LA    1,JHL
LOCATE      PUT   (1)                         LOCATE-MODE PUT MACRO
            L     5,0(15,3)                   BRANCH ON RC INDEX AND
            *                                 RCTABLE AS BASE
            BR    5
NORM        MVC   0(132,1),WORK
            .
            .
            .
            .
WORK        DS    CL132                       OUTPUT AREA FOR MOVE-
                                                MODE PUT MACRO
RCTABLE     DC    A(NORM)                     ADDR FOR PROCESSING
            *                                 AFTER RC OF 0
            DC    A(PAUSE)                    ADDR FOR PROCESSING
            *                                 AFTER RC OF 4
            DC    A(ERROR)                    ADDR FOR PROCESSING
                                              AFTER RC OF 8
```

Both the move-mode and locate-mode PUT macro instructions result in a type-1 linkage to the DOMSAM routine.

QGQE -- Queue Interrupt on Task

QGQE sets up a parameter list and calls CEAAF to queue the given GQE (containing the task interrupt information) on a task in interrupt priority sequence.

Name	Operation	Operand
[symbol]	QGQE	GQE [,INTRPT=]

GQE
generalized queue entry to be queued on the task.

Specified as: a system address, or a symbol defined on a fullword boundary that contains the address, of the GQE to be queued on the task.

INTRPT=
type of interrupt to be queued.

Specified as: a keyword, or register notation where the register contains the code for the desired interrupt type. Acceptable codes and keywords are as follows:

Keyword	Interrupt Type	Code
PROG	program check	(0)
PAGING	paging error	(1)
EXT	external	(2)
ASYNC/ATTN	asynchronous	(3)
TIMER	timer	(4)
SYNC/SYNICO	synchronous	(5)
VSS	VSS	(6)

CAUTIONS:
(1) The PSA dsect CHAPSA must have been copied by the module using this macro.
(2) The expansion of this macro is affected by the use of the CVT macro prior to the use of this macro.

Programming Notes:
(1) QGQE assumes that the GQE contains the address of the task which is the recepient of the interrupt (GQETSI).
(2) QGQE also assumes that the receiving task has been locked by the calling module.
(3) QGQE does no validity checking to make sure the GQE contains all the necessary interrupt information required for the specified interrupt type.

Example: To queue a synchronous I/O interrupt on a task, code:

```
L  R3,GQETSI
SETLOCK TSILOCK,12,01,LONG,,LOG     lock the task
+
QGQE  (R1),INTRPT=SYNCIO            queue the interrupt
+
OPENLOCK TSILOCK,,,LOG              then, unlock the task
```

QSVC -- Manipulate Resource Queue Entries SVC

The QSVC macro instruction, with its parameters set in registers 0, 1, 14, and 15, adds and deletes resource access entries from the supervisor resource access table.

Name	Operation	Operand
symbol	QSVC	

This macro instruction expects registers 0, 1, 14, and 15 to be set up as follows:

	for ENQ	for DEQ
register 0: byte 0		
bit 0	ENQ/DEQ flag 0 - ENQ	1 - DEQ
bit 1	lock type flag 0 - read lock 1 - write lock	DEQ type flag 0 - ECB 1 - ALL
bit 2	resource controller 0 - system control 1 - user control	--- --- ---
bit 3	VMADDR specified flag for ENQ and DEQ 0 - registers 14 and 15 contain NAME 1 - register 14 contains a VM address; 15=0	
bit 4	IMMED/INFINITE flag 0 - infinite wait 1 - immed	task to be purged 0 - issuing task 1 - taskid in reg 1
bit 5	resource owner flag for ENQ and DEQ 0 - system 1 - user	
byte 1	lock wait value 00 - IMMED/INFINITE 01 - SHORT 02 - MEDIUM 03 - LONG	--- --- --- --- ---
bytes 2-3	reserved	
register 1	ECB address or taskid	
registers 14-15	resource NAME or address	

Return codes: the following codes are returned in register 15 by the QSVC processor module:

Code	Meaning for ENQ	Meaning for DEQ
0	resource successfully accessed	DEQ successfully completed
4	resource in use; request queued except for IMMED	not used
8	error in parameters	error in parameters

Note: for code 8 for both ENQ and DEQ an error prompt message is in register 1.

Programming note: a QSVC for an ENQ IMMED request that cannot be satisfied will return the taskid of the task holding the resource in register 1.

RCALL — Call Another Supervisor Routine

RCALL sets up the linkages for calling another supervisor routine. RCALL loads Register 15 with the address of the entry point and does a BALR 14,15. The address is obtained from the CVT entry for the entry point.

Name	Operation	Operand
[symbol]	RCALL	EPT=

EPT=
name of the entry point of the routine to be called. The name must be in the CVT; otherwise, an assembly error will occur.

CAUTIONS:
(1) The PSA dsect CHAPSA must be copied in the assembly of the module using this macro.
(2) The expansion of this macro is affected by the use of the CVT macro prior to the use of this macro.

Example: To call CEATA1, code:

```
RCALL EPT=CEATA1
```

RDI -- Reset Drum/Disk Interlock (0)

Two lockbytes exist in the SYSTEM table to prevent interference between the supervisor and tasks when EREP records are read. One prevents supervisor to task interference, the second prevents task to task interference. The RDI macro resets the task to task lockbyte.

Name	Operation	Operand
[symbol]	RDI	

Note: There are no operands.

Execution: The task ID of the caller is matched against that of the interlock and the PSW condition code (contained in the XTSI) is set to one of the following:

0 The interlock is cleared.

1 The interlock was not cleared because the task ID of the issuing program did not match that of the TT interlock.

2 The interlock was not found set.

RECRDSTE -- Record Schedule Table Level Changes (0)

The RECRDSTE macro instruction records in a table (increments a counter), the movement of tasks from one schedule table level to another. This macro also maintains a list of the last 16 schedule table levels a task has been using in the task status index (TSI).

Name	Operation	Operand
[symbol]	RECRDSTE	base register,level register,work register,code

base register
specifies a register to be used as the base for the schedule table level change table.

Specified as: an absolute expression from 1 to 15.

level register
specifies the register that contains the new schedule table level.

Specified as: an absolute expression from 0 to 15.

work register
specifies a register to be used as a work register.

Specified as: an absolute expression from 1 to 15.

code
specifies the type of schedule table level change to be recorded.

Specified as:

Code	Meaning	Module
STXKZ1	TASK IN DELAY (TSEND)	CEAKZ
STXKZ2	TASK NOT IN DELAY (TSEND)	CEAKZ
STXKZ3	FREE VM LOCK (TSEND)	CEAKZ
STXNB	PAGE STEALING	CEANB
STXR2	PULSE SVC	CEAR2
STXR3	CHANGE SVC	CEAR3

Programming Note: The dsect CHASTX describes the schedule table level change table.

RELCORE -- Release Allocated Supervisor Work Space

RELCORE releases supervisor work space allocated by the GETCORE macro.

Name	Operation	Operand
[symbol]	RELCORE	ADDR,LGH,TYPE=

ADDR
address of the start of the area to be released.

Specified as: a system address that is on a 64-byte boundary. It may be given as register notation -- 1 is allowed -- or as a symbol defining a fullword which contains the address of the area to be released.

LGH
length of the area to be released.

Specified as: a register notation -- 0 is allowed -- or symbolic representation.

TYPE=
same as that specified in the GETCORE macro.

CAUTIONS:
(1) The PSA dsect CHAPSA must be copied by any module assembly using this macro.
(2) The expansion of this macro is affected by the prior use of the CVT macro in the assembly.

Examples: To release an area of storage, code:

```
          RELCORE WRKADD,WRKLGH

WRKADD DC A(0)       area for saving address of work space
WRKLGH DC H'128'     length of work space
```

RESET -- Reset Device Suppression Flag (R)

The RESET macro instruction cancels the effects of a previous PURGE macro instruction by resetting a device's suppression flag in the TSDL.

Name	Operation	Operands
[symbol]	RESET	[device number]

device number
specifies the symbolic device address of the device whose flag you wish reset.

Specified as: A one-to-four digit decimal number, either as an absolute expression or in register notation (2 through 12); or as: ALL if suppression flags of all devices are to be reset.

Default: It will be assumed that the issuer has placed the symbolic device address in register 0.

Execution: The resident supervisor clears the device suppression flag in the task symbolic device list for the specified device address.

Return Data: If the device is not contained in the task's symbolic device list (an error), the high-order bit of register 0 is set to one.

Example: Suppose you want to allow I/O operations to continue on symbolic device 25. You might write:

```
      LH   2,=Y(25)
GO    RESET  (2)
```

RESETIR -- Reset Immediate Report Flag (0)

The RESETIR system macro instruction is used to specify that the immediate report error statistics for call type 25 (see Test and Maintenance User's Guide) channel inboard and outboard failures on direct access paging devices are to be ignored for a specified device; they are not to be recorded in the system's error recording areas on the paging drum and no message is to be written to the system operator.

Name	Operation	Operand
[symbol]	RESETIR	symbolic device address

symbolic device address
specifies the particular direct access device whose entry in the direct access statistical data record (CHBPSD) is to be marked so that no immediate-report statistical data will be written on the paging drum.

Specified as: A one-to-three digit hexadecimal number, as an absolute expression, or, if the symbolic device address is first loaded into register 0, as: (0).

Execution: When RESETIR (SVC 223) is executed, the immediate report flag (PSDIR) in the system's direct access paging statistical data record (DSECT CHAPSD) is set off. This flag indicates to the system's error recording routines (see System Service Routines) that any errors of call type 25 that occur on a specified device should not be recorded in the error recording areas on the system's paging drum.

Programming Notes: RESETIR is issued in system code as part of the RESET IR job option of the VMEREP command (see Test and Maintenance User's Guide). That is, when a privilege class E system programmer, using VMEREP, elects the job option of RESET IR, the RESETIR system macro instruction is executed as part of the resultant processing. The result is to reset the Immediate Report flag (PSDIR) in CHBPSD, which inhibits all IRs for paging devices from being stored on the paging drum.

The VMEREP job options for printing out error reports scan type codes that have been recorded on the paging drum along with the error records. If the PSDIR flag had been off during all error incidents and the system programmer issues the VMEREP job option SEARCH XX25 at his terminal, there would be no error statistics on the drum from which a report could be generated. If the flag had previously been on, and some call type 25 errors did occur on a paging device, those statistics would be written out in an immediate report. The immediate report flag PSDIR can be set

on via the VMEREP job option of SETIR (see Test and Maintenance User's Guide).

Example: To turn the immediate report flag (PSDIR) off for a device whose symbolic device address is 2C, a system programmer might code:

```
     NAME   RESETIR   2C
```

RESUME -- Return to Calling Program (O)

The RESUME macro instruction restores registers and returns control to the calling program.

Name	Operation	Operand
[symbol]	RESUME	area,(first register[,last register]) [,RC=return code]

area
specifies the address at which the data to be restored is located.

Specified as: An RX address, or register notation. If register notation is used, the address must first be loaded into the specified register.

first register
specifies the first register to be restored from the specified area.

Specified as: A decimal number or an absolute expression. This number must be greater than 7 and less than 16.

last register
specifies the last register to be restored from the specified area. The restoration has the same wrap-around feature as the STM or LM instructions.

Specified as: A decimal number or an absolute expression.

Default: If this operand is omitted, only the first register is restored.

RC=
specifies a return code to be sent back to the calling routine.

Specified as: A decimal number or an absolute expression. This number must be less than 4092 and must be a multiple of four.

Default: No return code is sent.

RETRNR -- Load Saved Registers and Return

RETRNR standardizes supervisor linkages. RETRNR reloads the saved contents of a calling program's registers from the supervisor save area stack, sets up returning parameter registers and a return code, and returns to the calling program.

Name	Operation	Operand
[symbol]	RETRNR	RC=,PREG=,PSWMSK=

RC=
indicates the return code.

Specified as: a decimal number to be placed in Register 15, or specified as Register 15 which means that it contains the return code and is not to be restored.

PREG=
identifies the register(s) containing parameters/values which are to be returned to the caller in Register 0 and/or 1.

Specified as: one or two registers. If two registers are given, then the contents of the first will be placed in Register 0, and the contents of the second in Register 1. If only one register is given, its contents will be returned in Register 1. See examples 2 & 3.

PSWMSK=
specifies whether the system mask was saved and is to be restored from the saved area.

Specified as:
Y - the mask is assumed to have been saved in the 68th byte of the save area.
N - the mask was not saved.

Default: N.

CAUTIONS:
(1) The PSA dsect CHAPSA must be copied by any assembly using the RETRNR macro.
(2) RETRNR assumes that a standard system save was done. If not, results are unpredictable.

Programming Notes: The standard system save area is 72 bytes in length. The format of the save area is as follows:

Word 1 - pointer to the previous save area.
Words 2 to 17 - register save in the sequence 14-13.
Word 18 - before return, it contains previous system mask if saved; after return, it contains return code for debugging.

Examples:

(1) Normal return with a return code of 0,

```
RETRNR RC=0
```

(2) Normal return with a returning parameter and a code in Register 15,

```
RETRNR PREG=(5),RC=(15)
```

the returning value from Register 5 will be placed in Register 1 for return to the calling routine and register 15 will be unchanged.

(3) Returning with return values in Registers 7 & 3, and a return code of 8,

```
RETRNR PREG=(7,3),RC=8
```

Register 7's value will be returned in caller's Register 0, and Register 3's value will be returned in caller's Register 1. Register 15, on return, will contain a return code of 8.

RJELC -- Remote Job Entry Line Control (O)

The RJELC macro instruction causes the resident supervisor to execute channel programs that enable or disable the line associated with a particular remote device. It either enables the 2701 or 2703 channel control unit to receive input from a remote 2780 card reader or disables any transmission between them.

Name	Operation	Operand
[symbol]	RJELC	

Note: There are no operands. (See Initialization.)

Initialization: Before issuing RJELC, register 0 must be loaded with the symbolic device address of the remote device that is to be initialized, and register 1 must be loaded with an action code indicating whether the line to the device is to be enabled, disabled, or primed (i.e., reinitialized). The requirements for writing these parameters are indicated below:

Reg 0 - sda — Register 0 must contain the symbolic device address assigned during the system generation process. You may write: LA 0,X where X is the decimal number assigned to the particular device.

Reg 1 - codes

0 — Prime the line; reinitialize the specified line for data transfer (e.g., to regain line control after a 2780 time out condition).

1 — Enable the line; activate the specified line and initialize it for data transfer. This operation is invoked as a result of the operator issuing the ASNBD command.

2 — Disable the line; deactivate the specified line after either normal or abnormal job termination.

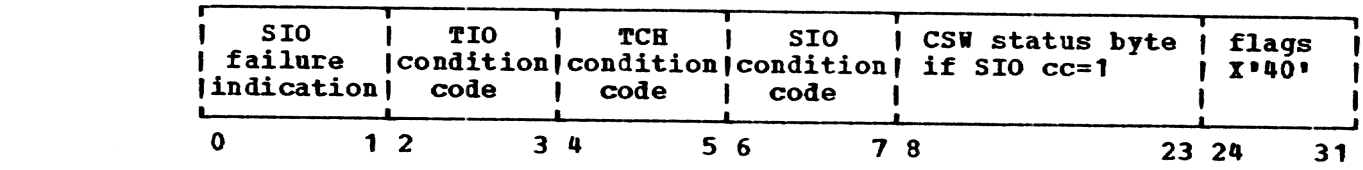
Execution: When the SVC 232 generated by RJELC is executed, the resident supervisor attempts to execute channel programs which perform the operation requested (enable, disable, or prime) on the line to the remote device specified. The line can be a dedicated or dial-up line. If a dedicated line is enabled, the line is prepared to receive input from the remote device immediately. If a dial-up line is enabled, the line is prepared for receiving the dial-in and then, once the user dials in, the dial-up line is prepared to receive input from the remote device.

Return Data: Registers 1 and 0 contain the following parameters:

Register 1 = Code	Meaning
0	SIO successful
4	SIO failed; also examine register 0
8	Path unavailable or invalid input
12	Path busy

When the return code in register 1 is 4:

Register 0 =



Where bits 0-1 If on, SIO failed (these bits not meaningful to macro execution).
2-3 See TIO assembler instruction.
4-5 See TCH assembler instruction.
6-7 See SIO assembler instruction.
8-23 See CSW status byte.
25 If on, control unit busy.
26 If on, expected interruption taken by other CPU.

Programming notes: In TSS, RJELC is issued in the BULKIO Initialization routine in response to the ASNBD command. Prior to executing the SVC, the Initialization routine defines (via a SIR macro instruction) an interruption processing routine to service any asynchronous interruptions from specified lines that occur during the SVC processing. The BULKIO Input Start routine is defined as the processor.

Example: The line to the remote device with the symbolic device address of 123 is to be enabled for receiving input.

```
        LA      0,123       LOAD SYMBOLIC DEVICE ADDRESS
        LA      1,1         SET REQUEST FOR ENABLING OPERATION
        RJELC
        SVC     232
```

RMDEV -- Remove Device From Task Symbolic Device List (R)

The RMDEV macro instruction removes an I/O device from your task's symbolic device list.

Name	Operation	Operand
[symbol]	RMDEV	[device]

device
specifies the symbolic device address of the device that you want removed from your task's symbolic device list (TSDL), and whether the DEVT flag is to be set off. Setting the flag off indicates that the issuer does not wish to control I/O when that device is again assigned by ADDEV. (See the CKALOC macro instruction for a further discussion of the DEVT flag.)

Specified as: Register notation (2 through 12). If the two's complement of the symbolic device address is used, the DEVT flag will also be set off.

Default: It will be assumed that the symbolic device address (or its two's complement) has been placed into register 0.

Execution: The resident supervisor reduces the ADDEV count in the task's symbolic device list by 1. If the count is reduced to 0, the device entry is removed from the task's symbolic device list.

Return Data: If the symbolic device address is not found in the task's symbolic device list, the supervisor sets the high-order bit of register 0 to 1.

Example: Suppose you want to remove symbolic device 46 from your symbolic device list; assuming no other part of your task had also added device 46, the ADDEV count for device 46 would be 1. You might write:

```
      LH    2,=Y(46)
GON   RMDEV  (2)
```

RMOVHLD -- Remove Page from Page Hold

RMOVHLD takes a VMA or a list of VMAs and decrements the page hold count of each page by 1.

Name	Operation	Operand
[symbol]	RMOVHLD	{VMA=|TCW=},TSI

VMA=
virtual memory address of a single page whose page hold count is to be decremented by 1.

Specified as: register notation (2 through 12 or 0), or a symbol defining a fullword containing the VMA.

TCW=
the address of a page list (for format, see dsect CHATCW).

Specified as: register notation (2 to 12), or a symbol defining a fullward containing the TCW address.

TSI=
address of the TSI for the task owning the page or pages. The task must have been locked by the calling routine.

Specified as: register notation (2 to 12 or 1), or a symbol defining a fullward containing the TSI address.

CAUTIONS:
(1) The PSA dsect CHAPSA must be copied by any assembly using the RMOVHLD macro.
(2) The use of the CVT macro prior to the use of the RMOVHLD macro affects the expansion of this macro.

Programming Notes: The macros GETPAG and RMOVHLD operate as a pair. The GETPAG macro is used to bring the virtual memory page into storage and to place the page in page hold, while the RMOVHLD macro is used to remove the virtual memory page from page hold when the routines are finished with the virtual memory page.

Examples:

(1) To remove a single page from page hold, code:

```
RMOVHLD VMA=(5),TSI=(2)
```

(2) To remove a list of pages from hold when the list is in TCW format, code:

RMOVHLD TCW=(5),TSI=(2)

ROPAGE -- Read Only Page Protection Flag Update (R)

The ROPAGE macro instruction is used to test, set, or reset the READ ONLY page bits in the external page tables.

Name	Operation	Operand
[symbol]	ROPAGE	VMA=,COUNT=,OPTION={TEST|SET|RESET}

VMA=
address of first page to be operated on.

Specified as: an RX address, or as register notation. Execution time is saved if register 1 is specified. If register notation is used, the address must first be loaded into the specified register.

Default: None.

COUNT=
number of pages to be operated on.

Specified as: an RX address, or as register notation (except 1). Execution time is saved if register 0 is specified. If register notation is used, the count must first be loaded into the specified register.

Default: none.

OPTION=
indicates whether the READ ONLY bits in the XPT (External Page Table) are to be tested, set, or reset.

Specified as: TEST, SET, or RESET.

Default: none.

Return codes: if OPTION=TEST, the following codes will be returned in register 15:

Code	Meaning
0	pages are not read only protected.
4	pages are read only protected.
8	pages are mixed; SET or RESET cannot be performed on this range of pages with only one issuance of the ROPAGE macro.
12	indicates an erroneous count parameter.
16	indicates an unassigned page.
20	indicates mixed shared and private pages.

Note: for return codes 16 and 20, the first erroneous address is returned in register 1.

If OPTION=RESET, a return code is set as above. If the return code equals 4, the READ ONLY protection bits were turned off.

If OPTION=SET, a return code is set as above. If the return code equals 0, the READ ONLY protection bits are now turned on.

RPRMPT -- Send Message to Task

RPRMPT allows a supervisor module to send a message to a task using the Virtual Memory Prompt facility. The RPRMPT mechanism allows up to five variable message inserts.

Name	Operation	Operand
[symbol]	RPRMPT	P1,P2,P3,P4,P5,MSGID=,TSI=,TSKID=

P1,P2,P3,P4,P5
the variable message inserts; the format for Px is:
(TYPE[,LGH],NAME)

TYPE
the converted display characteristic of the parameter.

Specified as: H - printable hexadecimal
D - printable decimal number
'null' - printable character string

LGH
designates the length of the field defined by NAME.

Specified as: a number, 1 to 15.

Default: the length attribute (L'NAME) of the field defined by NAME.

NAME
the name of a variable insert field to be converted, formated, and printed.

Specified as: any term, or register notation.

Note: Only NAME above may be given as other than a self-defining term.

MSGID=
address of an 8-byte character field containing the message ID of the message to be prompted.

Specified as: a system address, or a character string.

TSI=
the address of a TSI which may have been locked by the caller. (This parameter is used by RPRMPT to prevent an attempt to lock a task already locked by the caller.)

Specified as: an RX address or register notation

TSKID=
the ID of the task to receive the message.

Specified as: an RX address or register notation or task id number

Default: task 1 -- SYSOPER0

CAUTIONS:
(1) The PSA dsect CHAPSA must be copied and assigned a base register by

any assembly using this macro.
(2) The expansion of the RPRMPT macro is affected by the use of the CVT macro prior to the use of the RPRMPT macro.

Programming Note: For all variable insert parameters of type 'D' a left zero suppress is done.

Example: To send the following message to the operator

UNEXPECTED INTERRUPT: SDA=$,CSW=$2;

the RPRMPT macro would be coded like:

```
RPRMPT (H,,TCTSDA),(H,8,GQECSW),MSGID='CEATA001',TSI=(REG2),TSKID=1
```

WHERE TCTSDA is a halfword field which contains the SDA of the device and GQECSW is an 8-byte field containing the CSW from the interrupt being recorded. The macro assumes that the specified MSGID exists in either TSS SYSLIB SYSMLP or in the user's USERLIB SYSMLP.

RSEG -- Reserve Segment (O)

The RSEG macro instruction transfers control to the reserve segment program in the resident supervisor. An attempt will then be made to reserve the specified segment group.

Name	Operation	Operand
	RSEG	

Note: There are no operands.

Initialization: Before executing RSEG, the issuing program should have set up the following parameter area:

```
CHANSG   DSECT ,          COMMON NAMESEG PARAMETER LIST
         DS    0F
NSGSVC   DS    H          SVC
         DS    XL2        RESERVED

NSGRNA   DS    XL8        RESERVED SEGMENT GROUP NAME
NSGDNA   DS    XL8        DISCONNECTED SEGMENT GROUP NAME

NSGVMA   DS    A          VIRTUAL STORAGE ADDRESS OF SEG GROUP

NSGLNG   DS    H          LENGTH OF NAMED GROUP

NSGFLI   DS    XL1        INPUT FLAGS
NSGFLO   DS    XL1        OUTPUT FLAGS
NSGDNGM  EQU   X'80'      DNAME SPECIFIED
NSGRNGM  EQU   X'40'      RNAME SPECIFIED
NSGADGM  EQU   X'20'      ADDRESS SPECIFIED
NSGBNDM  EQU   X'10'      MODE=BOUND
NSGLNGM  EQU   X'06'      LENGTH SPECIFIED

         DS    F          RESERVED
NSGLTH   EQU   *-CHANSG   LENGTH OF PARAMETER LIST
```

RSEG must be the object of an execute instruction and be fullword aligned.

Execution: This macro instruction passes control to the resident supervisor module CEAP2 via SVC 180. An attempt will then be made to reserve the specified segment group.

Programming Note: The system programmer should use the RSVSEG macro instruction described in Assembler User Macro Instruction, GC28-2004.

RSPRV -- Restore Privilege (R)

The RSPRV macro instruction returns control to a caller who used type-3 linkage or to any non-privileged interruption routine dispatched by the task monitor.

Name	Operation	Operand
[symbol]	RSPRV	

Note: There are no operands.

Initialization: A DCLASS macro instruction with the USER option must be coded in a module prior to coding RSPRV. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding RSPRV must be issued with the USER option.

Execution: A task-SVC interruption is created to transfer control to the task monitor. After return from a non-privileged interruption routine, the task monitor restores all registers to their values at the time of the interruption and then scans for more interruption routines to dispatch. To complete type-3 linkage, the task monitor restores registers 2 through 14 to the values they contained when received from the privileged calling program. Registers 0, 1, and 15 are left unchanged (see "Linkage Conventions"). Control is then returned to the privileged program which invoked the type-3 linkage.

Example: Suppose you have written a type-3 program which has received control from the Leave-Privilege routine and is now ready to return control to the privileged calling program. You might write:

```
       DEPART  RSPRV
```

You could also write:

```
               BR 14
```

since register 14 is set to point to an SVC 120 by the Leave-Privilege routine. Return from a non-privileged interruption routine dispatched by the task monitor can also be coded either way.

RSSERR -- Indicate RSS Logic Error

RSSERR informs the RSS user of the occurrence of an RSS logic failure and saves error conditions at the time of the logic failure for easier debugging and recovery.

Name	Operation	Operand
[symbol]	RSSERR	SEV,CODE,ID

SEV
indicates the severity of the error.

Specified as: 1 - minor, restart should be successful
2 - major, results of any restart are unpredictable

CODE
represents a four-byte identification number

ID
represents the ID of the module discovering the logic failure

CAUTIONS:
RSSERR can only be used wihtin the RSS environment. The RSSERR issues an RSS SVC which may cause unpredictable results if RSS is not active. (2) Currently RSS error module CEHER stops execution on both minor and major RSS errors. To attempt RSS restart, press the cpu restart key twice.

Programming Notes: RSS will print/display the following message when an RSSERR is encountered:

...*****[MINOR|MAJOR][1] RSS ERROR 0000[2] FROM MODULE:module name[3]

...ERROR PSW:...TYPE-[PROGRAM|SVC][4]...error psw[5]...error code[6]

REGISTERS:[7] REGISTER0 REGISTER1 REGISTER2 REGISTER3
REGISTER4 REGISTER5 REGISTER6 REGISTER7
REGISTER8 REGISTER9 REGISTER10 REGISTER11
REGISTER12 REGISTER13 REGISTER14 REGISTER15

[1]MINOR|MAJOR - severity of detected error. Currently, recovery is the same; the RSS user must press the RESTART key on the CPU twice to reinitialize RSS.

[2]this field contains the error code from the RSSERR macro to help the programmer to determine the errror encountered.

[3]Module ID of the module issuing the error macro, or receiving the program check.

[4]PROGRAM|SVC is the type of error indication:
PROGRAM - error was an unexpected program check
SVC - error was a RSSERR macro call

[5]error psw - the PSW at the time of the error

[6]interrupt code - the interrupt code from the error.

[7]registers - the contents of the registers at the time of the error indicator.

RSVSEG -- Reserve Segment Group ((O)

This macro instruction is completely documented in the Assembler User Macro Instruction manual, except for one operand that is available only to a systems programmer. The definition and specification for only that one operand are given below, but for continuity, the metalanguage format that follows shows all the operands.

L-form

Name	Operation	Operand
Symbol	RSVSEG	[RNAME=,LENGTH=,RSTRCT=,] MF=L

E-form

Name	Operation	Operand
[symbol]	RSVSEG	[RNAME=,LENGTH=,ADDRESS=,RSTRCT=,] MF=(E,list)

Standard-form

Name	Operation	Operand
[symbol]	RSVSEG	[RNAME=,LENGTH=,ADDRESS=,RSTRCT=]

Note: All operands are keyword.

RSTRCT=
specifies whether system GETMAIN requests will be allowed in the reserved segment area when the area has been disconnected with the RELEAS=Y option specified in a DISCSEG or EXCSEG macro.

Specified as:

Y - system GETMAIN requests will not be allowed even though RELEAS=Y was specified in a DISCSEG or EXCSEG macro.

N - system GETMAIN requests will be allowed if RELEAS=Y was specified in a DISCSEG or EXCSEG macro.

This option is available to privileged class programs only. Refer to GETMAIN, DISCSEG, and EXCSEG macros for further information.

RTRN -- Return and Cleanup Task (R)

The RTRN macro instruction causes the system's command analyzer to cleanup a task prior to task termination. (Also see the RTRN command in Command System User's Guide.)

Name	Operation	Operand
[symbol]	RTRN	

Note: There are no operands.

Execution: A task-SVC interruption is created to transfer control to the task monitor. The task monitor in turn calls the Command System RTRN routine which resets the task as follows: all user SIRs are forgotten as are all user interruption routines and active user programs; the source list is reset to its initial state and all pending attentions

from the terminal are discarded; if a USATT had been given, the Command System regains control of attentions (just as if a CLATT had been issued); all AETD's are forgotten.

Example: If the program you caused to run is finished and you want to return control to the command analyzer for end-of-run processing, you might write:

```
        NAME  RTRN
```

RTTCTL -- Real Time Task Control (O)

The RTTCTL macro instruction provides four functions, which are defined below, for controlling real time tasks.

Name	Operation	Operand
[symbol]	RTTCTL	desired function

desired function
identifies the real time task control function.

Specified as:

INITIAL - initialize the task for subsequent delta time waits and turn on the real time flag.

FWAIT - setup for a future timer interrupt, turn on the real time flag, and place the task in AWAIT until the interrupt occurs.

DWAIT - setup for a timer interrupt, according to the base and delta times previously specified, and place the task in AWAIT until the interrupt occurs.

CLEAR - clear all time values and turn off the real time flag.

Default: none.

Initialization: for certain functions, registers 0, 1, and 15 must be loaded with time values in microseconds. The following table indicates the time values required:

code	registers 0:1	register 15
INITIAL	base time	delta time
FWAIT	future time	
DWAIT		
CLEAR		

A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding RTTCTL. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding RTTCTL must be issued with a PRIVILEGED option.

CAUTION: the time specified in registers 0:1 must correspond to a point in time greater than the current time of day clock. The delta time specified in register 15 must not be zero. Registers 0, 1, and 15 may be modified during execution of this macro instruction. The real time flag turned on by this macro instruction assumes that page stealing is specified in the schedule table levels used by this task.

Execution: the FWAIT and DWAIT codes cause an entry to be created in the real time interrupt queue for this task. For the FWAIT code, the interrupt time is taken directly from registers 0:1. For the DWAIT code, the interrupt time is

(BASE TIME) + N*(DELTA TIME)

where N is the number of times the DWAIT code has been issued since the last INITIAL code.

When the time interrupt occurs the task is moved to the active list if necessary. The deadline dispatch time is computed and stored in the task's TSI for processing by the dispatcher. This deadline dispatch time is formed by adding the timer interrupt time to a dispatch delay time speacified in the real time task control SVC processor. The dispatcher will (in a multi-CPU configuration) stop the other CPU from executing in a lower priority task so that a higher priority task with a deadline dispatch time may be placed in execution.

The real time flag turned on by this macro instruction disables the time slice end function of purging pages. This tends to keep the task's pages in real storage for extended periods of time.

Return data: one of the following return codes is placed in register 15 upon completion of execution for this macro instruction:

code	meaning
0	no error
4	invalid delta time
8	invalid base of future time
12	not used

Programming notes: all times specified in this macro instruction must be in microseconds. The current time in microseconds may be obtained by using the STCK instruction and shifting the result to the right 12 bits.

Example: assume you wish your task to be placed in execution exactly every tenth of a second beginning about 10 seconds in the future. You might code:

```
        STCK    CLOCK           GET TIME + ABOUT 10 SECONDS
        LM      0,1,CLOCK       *
        AL      0,=F'10'        *
        SRDL    0,12            *
        L       15,=F'100000'   GET .1 SECONDS IN MICROSECONDS
        RTTCTL  INITIAL         INITIALIZE REAL TIME TASK
RTM1    DS      0H
        RTTCTL  DWAIT           WAIT FOR TIMER
          .                     |
          .                      >PROCESS
          .                     |
        B       RTM1            BRANCH TO WAIT
CLOCK   DS      D
```

SAMPLE -- Sample Statistical Recording Fields (O)

The SAMPLE macro instruction moves system status information, maintained in main storage, into virtual storage.

Name	Operation	Operand
[symbol]	SAMPLE	

Note: There are no operands.

Initialization: Before executing the SAMPLE macro instruction, one of the following codes must be set up in general register 0:

Code	Meaning
4	SAMPLE task paging activity
6	SAMPLE system statistics
8	SAMPLE task schedule table history

When using codes 4 or 8, the SAMPLE macro instruction may be executed in line with other program instructions. All data are required in the floating point registers.

When using code 6, SAMPLE must be the object of an EXECUTE instruction. SVC 193, which SAMPLE produces, must occupy the first half word within the virtual storage page where the statistical data is to be recorded. Also, the second double word, bytes 8 through 15, of the virtual storage page must contain one of the following 8 byte character strings:

Character String	Meaning
'SYSTEM '	SAMPLE system parameters
'GLOBAL '	SAMPLE global system statistics
'LOCAL '	SAMPLE memory and task statistics

CAUTION: Use of the SAMPLE macro instruction (SVC 193) is restricted to tasks having system programmer authority (0 or P).

Execution: On execution of the SAMPLE macro instruction, statistics maintained in main storage system status table (CHBSST) and several other system tables, are moved, depending upon the code in register 0, into either the virtual storage page containing the SAMPLE SVC or the floating point registers.

The SVC routine checks whether the data area in the receiving page is equal to or greater than the data bytes to be transferred. If it is not, data transfer is halted when the area becomes full.

Return Data: After execution of the SAMPLE macro instruction, one of the following return codes is placed in general register 15:

Return Code	Meaning
0	no error
4	invalid code in register 0
8	SAMPLE SVC not in first halfword of virtual storage page

Code 4 in general register 0 provides the following task related counters in the floating point registers:

F. P. Register	Contents
0	private drum reads
1	shared drum reads
2	private disk reads
3	shared disk reads
4	reclaimed pages
5	SVCs issued
6	relocation exceptions
7	auxiliary pages

Code 6 in general register 0 transfers data to a virtual storage page. DSECTs are provided to cover the three types of data transfers:

Data	DSECT
system parameters	CHAPXS
GLOBAL SYSTEM STATISTICS	CHASTV
MEMORY AND TASK STATISTICS	CHATSX

Code 8 in general register 0 produces the following task related data in the floating point registers:

F. P. Register	Contents
0	task time in milliseconds
1	count of time slice ends
2-5	last 16 schedule table levels used

Note: All data returned in floating point registers are in fixed point format.

Programming Note: Additional information pertaining to the use of SAMPLE can be found under "Evaluation of System Status Statistics."

Example: A system programmer wants to extract the current system parameters (described by CHAPXS). He might write the following:

```
         .
         .
         .
         GETMAIN  PAGE,LV=1            GET PAGE FOR DATA
         LR       3,1
         USING    CHAPXS,3             COVER PAGE
         MVC      PXSSVC,SAMP          INITIALIZE PAGE
         MVC      PXSNAME,=CL8'SYSTEM' INITIALIZE PAGE
         LA       0,6                  SAMPLE CODE
         EX       0,PXSSVC             EXECUTE SVC
         .
         .
         .
SAMP     SAMPLE
```

SAVER -- Supervisor Standard SAVE Function

SAVER standardizes the supervisor function of saving the calling modules's environment and registers. Upon exit, Register 13 points to the save area containing the calling module's saved registers.

Name	Operation	Operand
[symbol]	SAVER	,LGH

Note: The first operand is ignored; it is only included to maintain the correct parameter displacement.

LGH
the length of a work area to be allocated from the save area stack. The address of the beginning of the work area will be returned in Register 14.

Specified as: an integer number 1 - 256, or register notation

CAUTION: The PSA dsect CHAPSA must be copied by any assembly using the SAVER macro.

Programming Notes: The standard TSS supervisor SAVE area is 72 bytes in length. The registers are saved in the sequence 14-13. The format of the SAVE area is as follows:

Word 1 - pointer to previous save area used by the calling module.
Words 2-17 - contents of the calling module's registers in the sequence 14-13.
Word 18 - used as a work area.

Each CPU in a TSS configuration is assigned its own save area stack which is pointed to by the CPUs PSA. Within the PSA is another pointer to the next available word in the save area stack. As a program is called and does a SAVER, GETWORK, and RETRNR, this PSA pointer is updated to reflect the next available area. When a SAVER is done the current value of the PSA pointer is saved in the first word of the save area. Then as the GETWORKs are issued the pointer is updated to reflect the allocated areas. Finally when the module issues a RETRNR to exit, the saved PSA pointer is restored to the PSA and any allocated work areas are made available for reuse.

The PSA save area pointer is reinitialized to the top of the stack on each pass through CEAJQS. It is the responsibility of any module exiting out of line, expecting to be returned to later to resume execution, to save the save area stack, allocate a new stack, and when resuming execution, to release the current stack, and restore the saved stack.

Example: To save registers and allocate a 64-byte work area, code:

```
SAVER   ,64
```

After execution, Register 13 will point to the start of the save area and Register 14 to the start of the work area.

SCHED - Schedule Table Entry (R)

The SCHED macro instruction is used to communicate a schedule table entry (STE) to the supervisor for use in scheduling a task. The information is placed in the task's TSI.

Name	Operation	Operand
[symbol]	SCHED	

Note: There are no operands.

Execution: The SCHED macro instruction forms a value made up of the user's schedule table index (which identifies the user's priority) and the type of task (conversational or batch). This value is placed in register 15 and is passed to the resident supervisor by means of the

CHANGE macro instruction, using the STE code in that macro instruction's operand.

Return Data: See the CHANGE macro instruction.

Example: Suppose that a conversational task is switched to the background and you want to establish a new schedule table entry for it. To communicate a new STE to the supervisor, you write:

```
NAME      SCHED
```

SCRTSI -- Special Create Task Status Index (R)

The SCRTSI macro instruction allocates storage for and initializes a TSI regardless of the number of TSIs already defined. (CRTSI provides the same service, but places a limit on the number of TSIs that can exist at one time.)

Name	Operation	Operand
[symbol]	SCRTSI	[taskid]

taskid
specifies the taskid to be assigned to the TSI that is to be created.

Specified as: an RX address.

Default: none.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding SCRTSI. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding SCRTSI must be issued with the PRIVILEGED option.

Execution: A new task status index is created regardless of the number of TSIs in existence. If the system TSI limit has been reached, it is incremented. After the TSI has been created, the limit is restored. For an explanation of how the TSI is initialized, see the CRTSI macro instruction.

Return Data: The id of the new task is returned in register 0.

Example: If you want to create a TSI, you might write:

```
ABC    SCRTSI
```

SETAE -- Set Asynchronous Entry (R)

The SETAE macro instruction indicates which task is to receive asynchronous interruptions from a specified device.

Name	Operation	Operand
[symbol]	SETAE	[device number][,task]

device number
specifies the symbolic device address of a the device to be set.

Specified as: Register notation (2 through 12).

Default: It is assumed that the issuer has loaded the symbolic device address in register 1.

task
specifies the identification of the task to which the device is to be assigned, whether the device is to be made unassigned (restored to a neutral state), and whether the DEVT flag is to be cleared.

Specified as: Register notation (2 through 12), expressing the task identification (taskid), if the device is to be assigned. The value must be zero if the device is to be made unassigned. If X'FFFFFFFF' is specified in register notation, the device is made unassigned and the DEVT flag is cleared (see the description of the CKALOC macro instruction for a discussion of the DEVT flag).

Default: The device is made unassigned.

Execution: The entry in the asynchronous device group table (CHBADT) for the specified symbolic device address is set to point to the task status index of the task referred to by the task ID. If no task operand is specified, the symbolic device is marked unassigned. An entry is also placed in the TSDL of the new task.

Example: Suppose you want attention interruptions received from device 124 to be processed by the task, having task identification 233. You might write:

```
        LH   2,=Y(124)
        LH   3,=Y(233)
EST     SETAE  (2),(3)
```

SETCTL -- Set Control Registers (R)

The SETCTL macro instruction allows you to set selected task control registers.

Name	Operation	Operand
[symbol]	SETCTL	[field-{MCMASK|PERMASK|PERGR|PERADDR|(15)}]

field
designates the control register function you wish to set and may be written:

```
MCMASK  - set monitor call mask CR 8 bits 16-31
PERMASK - set program event recording mask CR 9 bits 0-3.
PERGR   - set PER general register mask CR 9 bits 16-31.
PERADDR - set PER address range CR 10-11.
```

If you choose to write register notation, you must select the proper value from the list below and place it in register 15 before issuing the macro instruction.

Code	Value
MCMASK	1
PERMASK	2
PERGR	3
PERADDR	4

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding SETCTL. If more than one

DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding SETCTL must have been issued with the PRIVILEGED option.

Execution: The contents of registers 0 and 1 are placed in the control register save area of the XTSI corresponding to the code contained in register 15. The control registers are subsequently loaded from this save area before the task is placed in execution. The number of bytes to be inserted into the control register save area depends on the code:

Code	Implied Length (bytes)
1	2
2	1
3	2
4	8

Regardless of the length, the bytes are always to be right-justified in registers 0 and 1.

Example: Suppose you wish to enable monitor call class 2. This is controlled by turning on bit 18 in control register 8. SETCTL replaces bit positions 16-31 in control register 8, so an XTRCTL macro should be used to inspect the current bit settings in control register 8. Then you might write:

```
         LA       1,288
ENAB2    SETCTL   MCMASK
```

SETIR -- Set Immediate Report Flag (0)

The SETIR system macro instruction issued to specify that the immediate report error statistics (call type 25 channel inboard and outboard failures on direct access paging devices) for a specified device are to be recorded in the system's error recording areas on the paging drum. (Also see Test and Maintenance User's Guide).

Name	Operation	Operand
[symbol]	SETIR	symbolic device address

symbolic device address
specifies the direct access device whose entry in the paging statistical data table (CHBPSD) is to be marked for immediate-report statistical output to the paging drum.

Specified as: The hexadecimal equivalent of the symbolic device address (usually expressed in decimal) or, if the symbolic device address is first loaded into register 0, as (0). (This may be done by a Load Address instruction, expressing the symbolic device address in decimal.)

Execution: When SVC 223, generated by SETIR, is executed, the immediate report flag (PSDIR) in the system's direct access paging statistical data record (DSECT CHAPSD) is set on. This flag subsequently informs the system's error recording routines (see System Service Routines) that call type 25 errors that occur on a specified device should be recorded in the error recording areas on the system's paging drum.

Programming Notes: SETIR is issued in system code as part of the SETIR job option processing of the VMEREP command (see Test and Maintenance User's Guide). Thus, when a privilege class E user (system monitor), using VMEREP, elects the job option of SET IR, the SETIR macro instruction is executed as part of the processing that results.

The VMEREP job options for printing error reports use statistics that have been recorded on the paging drum. If the PSDIR flag is on, and the user issues the VMEREP job option SEARCH XX25 at his terminal, then a report based on any statistics that may have been recorded in the drum (the immediate report) would be printed at his SYSOUT device, listing all devices that had errors of call type 25 (and each such error on the device). The format for these immediate reports is described in the Test and Maintenance User's Guide.

When the PSDIR flag is off, call type 25 errors on the associated device are not recorded on the drum; in this case no immediate report statistics are available (unless the PSDIR flag had previously been on) when the privilege class E user elected the job option SEARCH XX25. The VMEREP job option RESET IR can be used to set the PSDIR flag off (see also the description of the RESETIR macro instruction).

Example: To turn the immediate report flag (PSDIR) on for a device whose symbolic device address is hexadecimal 19, a system programmer might code:

```
NAME SETIR 19
```

SETLOCK -- Set a Resident Supervisor Lock Byte (O)

The SETLOCK macro instruction locks a designated table or function in the resident supervisor, protecting the table or function in an MP or AP (two processor) environment.

Name	Operation	Operand
[symbol]	SETLOCK	[count register],lockbyte,[module id], [error number],{IMMEDIATE|SHORT|LONG},action, [exit address],[{LOG|NOLOG}]

count register
: specifies a register to be used for counting time increments. This operand is required if SHORT or LONG operands are specified; it will be disregarded if the IMMEDIATE operand is specified.

  Specified as: An absolute expression from 0 to 15.

lockbyte
: specifies the address of a location that is to be set to X'FF' using the TS instruction.

  Specified as: A symbolic address.

module id
: specifies the module trying to set the lock byte (the module in which the SETLOCK is issued). This operand will be used in generating a SYSER message in the event the attempt to set the lock byte is unsuccessful. This operand is required if the SHORT or LONG operand is specified; it is ignored if the IMMEDIATE operand is specified.

  Specified as: A two-digit decimal number.

error number
: specifies this particular SETLOCK macro instruction within a module in which more than one SYSER is issued. This operand is used in generating a SYSER message in the event the attempt to set the lock byte is unsuccessful. This operand is required if the SHORT or

LONG operands are specified; it will be ignored if the IMMEDIATE operand is specified.

Specified as: A two-digit decimal number.

IMMEDIATE|SHORT|LONG
specifies how long to keep trying to set the lock in the event it is already locked.

Specified as:

IMMEDIATE	(only try to set it once)
SHORT	(keep trying until a length of time has elapsed)
LONG	(means the same as SHORT)

action
specifies what to do if the attempt to set the lock fails.

Specified as: A symbolic address to which to branch if the IMMEDIATE operand was specified.

exit address
specifies where to branch to if the lock was set or if a SYSER occurred.

Specified as: A symbolic address.

Default: The program continues at the next sequential instruction.

LOG|NOLOG
specifies whether the address of this SETLOCK macro instruction is to be logged in the logging field associated with the lock byte.

Specified as: LOG or NOLOG

Default: NOLOG

CAUTION: This macro instruction can only be issued within modules composing the resident supervisor.

SETSYS -- Set System Table Field (R)

The SETSYS macro instruction sets or alters one of a selected set of system table fields.

Name	Operation	Operand
[symbol]	SETSYS	[field]

field
designates the system table field you want to set or alter.

Specified as: TASKINIT

You may also use register notation by selecting the proper value from the list below and placing it in register 15 before you issue the macro instruction.

Code	Value
TASKINIT	3

Default: It is assumed that the issuer has placed a value in register 15.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding SETSYS. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding SETSYS must be issued with the PRIVILEGED option.

Execution: The contents of registers 0 and 1 are placed in the system table field corresponding to the code contained in register 15. The number of bytes to be inserted into the system table depends on the code:

Code	Implied Length (bytes)
3	1

Example: Suppose you want to inhibit the initiation of any more tasks. This is controlled by a flag byte in the system table called SYSTI; the appropriate bit mask is called SYSTIM. The task initiation bit is bit 2 of the SYSTI byte. SETSYS replaces the entire flag byte, so an XTRSYS should be used to extract the flag byte. Then you might write:

```
DOUG    SETSYS    TASKINIT
```

SETTIMER -- Set Realtime Interval From Resident Programs (S)

The SETTIMER macro instruction sets a realtime interval after which the system generates a timer interruption and passes control to a specified routine.

Name	Operation	Operand
[symbol]	SETTIMER	time ,return parameter interrupt routine entry point ,option

time
specifies the length of time in microseconds after which a realtime interruption will be generated.

Specified as: A decimal number, or, if the decimal number is first placed in one or two registers, in register notation. In register notation, if the number is too large for one register, it may be loaded into two contiguous registers and they may be specified as, for instance: (3,4).

return parameter
a value to be returned when the realtime interval has elapsed

Specified as: Register notation (2 through 15).

interrupt routine entry point
specifies the routine that is to receive control when the specified timer interruption occurs.

Specified as: An entry point name, or, if the address of the routine is first loaded into a register, in register notation (2 through 15).

option
specifies the type of value given by the first operand (time)

Specified as: ABS. If this operand is omitted or not ABS, the time value in the first operand is an interval. If ABS is coded

time is an absolute time when the interrupt is wanted. For example, a time may be calculated for the next hour, shift,day, etc.

Initialization: All modules using the SETTIMER macro instruction must have a copy of the DSECT CHAPSA and a USING CHAPSA statement.

CAUTION: If the second operand is missing, a comma must be included to indicate its absence.

Execution: If either the time or routine operand is not present, an error is indicated. Control is passed to the Set Realtime Interruption routine. The realtime interruption is eventually placed on the realtime interval queue (CHARTI) in the field RTITIME. Control is returned to the user with a return code set in register 15.

Return Data: Register 15 is set to 08 -- normal return.

If the program using SETTIMER wants to replace an existing interruption with a new one (new time value), the programmer must use the same routine and TSI addresses as were used by SETTIMER when it first placed the interruption on the queue.

The CANCL macro instruction can be used to cancel a programmed realtime interval previously established by a SETTIMER macro instruction.

Example: In EX1, below, a realtime interruption is desired 100 microseconds from the present time. In EX2, a realtime interruption is desired three minutes from the present time (that is, 180000000 microseconds). In EX3, an absolute time has been calculated in Registers 4 and 5.

```
EX1     L           8,TSIADD
        SETTIMER    100,(8),CEAMA
                    or
        SETTIMER    100,,CEAMA

EX2     LR          2,TSIADDR
        SETTIMER    180000000,(2),CEAMA

EX3     SETTIMER    (4,5),(2),CEAMA
```

SETTR -- Set Real Time Interval (0)

SETTR sets a time limit, in terms of a real time, on the execution of your task.

Name	Operation	Operand
[symbol]	SETTR	

Note: There are no operands.

Initialization: You must preload registers 0 and 1 with the time limit in microseconds.

A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding SETTR. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding SETTR must be issued with the PRIVILEGED option.

Execution: An entry is created in CHBRTI. If there was a previous entry for this task, it will be removed.

Return Data: Register 15 is set to 08, meaning normal return

SETTU — Set User Timer (R)

The SETTU macro instruction sets the user timer field in the XTSI, thereby limiting your task's execution time.

Name	Operation	Operand
[symbol]	SETTU	[time]

time
specifies the time duration in milliseconds that you want placed in the user timer field.

Specified as: A decimal number from 0 to 55364812 or, if the number is first placed in a register, in register notation (1 through 12).

Default: It is assumed that the issuer has placed the time duration in register 1.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding SETTU. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding SETTU must be issued with the PRIVILEGED option.

Execution: The quantity contained in register 1 is converted to microseconds and stored in the extended task status index field called user timer value (XTSUTI).

Example: Assume that register 5 contains the number of milliseconds to which you'd like to set the user timer. You might write:

```
NAME   SETTU    (5)
```

SETUP — Set Up Task Status Index Field (R)

The SETUP macro instruction permits you to alter or set the contents of a selected field in the TSI.

Name	Operation	Operand
[symbol]	SETUP	[field][,register]

field
specifies the field you want to set or alter.

Specified as: One of the codes described below, or, if a value corresponding to one of the codes (also shown below) is first placed in register 15, as (15).

USERID - set the user identification field
SYSIN - set the input data set location field
SYSOUT - set the output data set location field
SOPRIV - operator/(combined with privilege class-E)
system programmer privilege
SPPRIV - system programmer, nonprivileged
UPRIV - user

CONV - set the conversational task flag
ITMFLG - set the intertask message flag
XPR - set the external priority flag
AUTH - set the privilege field
MAV - set the maximum auxiliary storage field

Field	Value
USERID	1
SYSIN	3
SYSOUT	4
SOPRIV	6
SPPRIV	7
UPRIV	9
CONV	10
ITMFLG	12
XPR	13
AUTH	14
MAV	16

register
designates the even-odd register pair in which you have placed the information you want put into the specified TSI field.

Specified as: The odd register, expressed as an absolute expression or register notation.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding SETUP. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding SETUP must be issued with the PRIVILEGED option.

Execution: From one to eight bytes of registers 0 and 1 are inserted into the task status index field specified by the low-order byte of register 15. The number of bytes to be inserted depends on the field specified.

Field	Code	Implied length (bytes)
USERID	1	8
SYSIN	3	2
SYSOUT	4	2
SOPRIV	6	1
SPPRIV	7	1
UPRIV	9	1
CONV	10	1
ITMFLG	12	1
XPR	13	2
AUTH	14	1
MAV	16	2

Example: Assume that registers 12 and 13 contain an eight-character user identification. You might write:

```
TEST    SETUP    USERID,(13)
```

SETUR -- Set Up Unit Record Device (R)

The SETUR macro instruction specifies the configuration for online printers and card punches.

Name	Operation	Operand
[symbol]	SETUR	{dcb address[,setup]|parameter

dcb address
specifies the address of the data control block opened for processing a data set on a printer or card punch.

Specified as: A relocatable expression or register notation.

Default: None

setup
specifies the address of the desired form number for the punch. For printers, it specifies the name of the default region in the SYSUCS data set from which all printer defaults (FCB, CHAIN/TRAIN, etc.) may be found.

Specified as: one to six alphameric characters

Default: PAPER.

parameter pointer
specifies the address of a parameter list (which is defined by the CHASUR DSECT) which contains the exact specifications for a printer setup. This parameter list is in the following format:

```
SURORG   DS    0F
SURDCB   DS    A         ADDR OF MSAM DCB
SURCHARS DS    0C        START OF CHARS TO LOAD
SURCHAR1 DS    CL6       REG NAME OF 1ST CAT ENTRY
SURCHAR2 DS    CL6       REG NAME OF 2ND CAT ENTRY
SURCHAR3 DS    CL6       REG NAME OF 3RD CAT ENTRY
SURCHAR4 DS    CL6       REG NAME OF 4TH CAT ENTRY
SURFCB   DS    CL6       FCB REG NAME TO LOAD BY
SURDSN   DS    CL44      FQN OF SYSUCS DS TO USE FOR LOAD
SURCPDSN DS    CL44      FQN OF COPY MOD DS
SURBURST DS    CL1       Y! FOR BTS
SURPAPER DS    CL10      PAPER TO USE      ** NET **
SURCOPYG DS    X         NO COPIES OF DS
SURCOPY  DS    XL8       NO COPIES OF PAGE (ONLY 1ST BYTE USED)
SURFLASH DS    CL8       NAME OF FLASH IMAGE
SURFCNT  DS    X         COUNT OF COPIES TO FLASH
SURFORM  DS    CL6       NAME OF DEFAULT REGION IN UCS
SURVID   DS    CL6       VID THIS JOB
SURFLG   DS    X         FLAG BYTE
SURFLSH  EQU   SURFLG    0=FLASH MAY OR MAY NOT BE REQD
SURFLSHM EQU   X'80'     1=DO NOT FLASH
SURDFLT  EQU   SURFLG    0=CHARS PARM FILLED IN BY PRINT CMND
SURDFLTM EQU   X'40'     1=CHARS FILLED IN VIA DEFAULT
SURVID2  DS    CL6       VERSION ID OF COPY MOD DS
SURLEND  EQU   *
SURL     EQU   *-CHASUR  CURRENTLY USED LENGTH
         DS    2X        USED FOR ALIGNMENT (SPARE)
         DS    21F       USED FOR ALIGNMENT (SPARE)
SURLEN   EQU   *-CHASUR  LEN OF TABLE
```

Programming Notes: To ensure a valid setup, the SETUR macro instruction should be issued before any I/O operations are directed to a printer or punch. This is done by issuing SETUR immediately after opening a data set or after the FINISH macro instruction is executed and the I/O operation completed.

Card Punch: The setup for a card punch is described by the form number of the card that the operator is to load into the punch-feed hopper of the punch. This form number is an installation-defined constant. When the macro instruction is executed, the SETUR routine determines which form is mounted in the punch (the currently mounted form number -- or zeros if the DCB was just opened -- is stored for each device in the SDAT). If the desired form is already mounted, control is returned to the invoking routine with a return code of 0. If the form is not mounted, a message is written to the operator (WTO) to mount the desired form number (6-byte parameter), and to ready the punch. A return code of 4 is provided to the calling task. When the operator indicates that the punch is properly loaded, by causing a not-ready to ready interruption, the SDAT is changed to reflect the new form number. On the next call to SETUR, control is returned to the invoking routine with a return code of 0.

Printer: if the 'dcb address,setup' form of this macro is used, the value specified for the setup parameter is used as the index into the SYSUCS dataset from which printer setup defaults are obtained. The default region of the SYSUCS dataset must specify FCB, PAPER, and print train requirements.

When the 'parameter pointer' form of the macro is used, SETUR will fill in any missing defaults based upon the value specified in the SURFORM value in the parameter list. In either case, should a required parameter not be filled in, SETUR will issue an appropriate return code.

Execution: The SETUR macro instruction returns a code in register 15 indicating the manner in which the SETUR call was completed. All return codes are defined in Figure 35.

Return Code	Meaning
0	Operation completed successfully.
4	Operation not complete; SETUR macro instruction should be reissued.
8	Unrecoverable I/O error occurred while attempting to load the device.
12	User software error.
16	System software error.
20	RJE disconnect error.
24	Job cancelled.
28	Page backup requested.
	Note: for return codes 12 and 16, register 1 will point to a prompt parameter list indicating the exact cause of the error.

Figure 35. Return codes for the SETUR macro instruction

When the SETUR macro instruction is executed, the routine determines if the present configuration of the printer, specified in the SUR TABLE, pointed to by the SDAT, is the configuration requested for this SETUR call. If the form, carriage tape (FCB) chain/train, etc., are present, control is returned to the invoking routine. If the desired configuration is not present, the system acts to achieve the desired configuration.

SETUR uses the SYSUCS data set to build the necessary blocks to load a printer configuration. The SYSUCS data set used may or may not be user specified. If defaulted, SETUR uses the system owned SYSUCS data set TSS*****.SYSUCS(0); this data set contains all the information needed to load the 1403, 3211, and 3800 printers.

SYSUCS: this data set is a region data set consisting of 4 basic regions. Each region name is 8 bytes long, the first two bytes of which are predefined by the system. They are as follows:

1. CTXXXX -- character arrangement table region. This region contains the information needed to load the USCB in the 3211 and 1403 printers, and the translate tables and WCGMs in the 3800 printer. For the 3800 printer, it may also contain the name(s) of graphic modification regions. A maximum of 12 names may be specified.

2. FBXXXXXX -- format control buffer region. This region contains the information needed to indicate which density and carriage control tape are needed for the 1403 printer, and the FCB specification and density settings for the 3211 and 3800 printers.

3. GPXXXXXX -- graphic modification region. This region the picture images needed to built graphic modifications for the 3800 printer. All the standard IBM graphic modifications are in TSS*****.SYSGRAPH(0).

4. standard setup region. This region contains the default information needed for a standard printer setup. It is also used to backfill any setup information required but not specified.

Example 1: the example that follows contains the information needed to load the 3211 and 3280 printers with the P11 chain/train configuration. This is indicated by the DEVICE=3211/3800,NAME=P11 statements. The load information immediately follows this statement(s). In the case of the 3211 it is the chain/train image. For the 3800 it is the translate table followed by the WCGM ID. This example does not define the P11 train image for the 1403. However, it does indicate where this information may be found. The statement DEVICE=1403,SEE=(PN,1403) indicates the P11 compatible 1403 chain/train image may be found in the region CTPN of the SYSUCS data set.

```
CTP11    0000100 DEVICE=1403,SEE=(PN,1403)
CTP11    0000200 DEVICE=3211,NAME=P11
CTP11    0000300 1'BDJL-5K*C(NA@=E0?)S¬#R>V92"68<XYT|GF%H._UO7/P3WMIQ,4
CTP11    0000350 1'BDJL-5K*C(NO$=E:#)SA&RZV9+G682;YT<XF%H._UO7/P3WMIQ,4
CTP11    0000400 1'BDJL-5K*C(NA@=E0?)S¬#R>V92"68<XYT|GF%H._UO7/P3WMIQ,4
CTP11    0000450 1'BDJL-5K*C(NO$=E:#)SA&RZV9+G682;YT<XF%H._UO7/P3WMIQ,4
CTP11    0000500 1'BDJL-5K*C(NA@=E0?)S¬#R>V92"68<XYT|GF%H._UO7/P3WMIQ,4
CTP11    0000550 1'BDJL-5K*C(NO$=E:#)SA&RZV9+G682;YT<XF%H._UO7/P3WMIQ,4
CTP11    0000600 1'BDJL-5K*C(NA@=E0?)S¬#R>V92"68<XYT|GF%H._UO7/P3WMIQ,4
CTP11    0000650 1'BDJL-5K*C(NO$=E:#)SA&RZV9+G682;YT<XF%H._UO7/P3WMIQ,4
CTP11    0000700 END
CTP11    0001000 DEVICE=3800,NAME=P11
CTP11    0001100
CTP11    0001200     TRANSLATE TABLE
CTP11    0001300
CTP11    0001400 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
CTP11    0001500 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
CTP11    0001600 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
CTP11    0001700 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
CTP11    0001800 00FFFFFFFFFFFFFFFFFFFF0B0C0D0E0F
CTP11    0001900 10FFFFFFFFFFFFFFFFFFFF1B1C1D1E1F
CTP11    0002000 2021FFFFFFFFFFFFFFFFFF2B2C2D2E2F
CTP11    0002100 FFFFFFFFFFFFFFFFFFFF3A3B3C3D3E3F
CTP11    0002200 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
CTP11    0002300 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
CTP11    0002400 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
CTP11    0002500 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
CTP11    0002600 FF0102030405060708 09FFFFFFFFFFFF
CTP11    0002700 FF1112131415161718 19FFFFFFFFFFFF
CTP11    0002800 FFFF2223242526272829FFFFFFFFFFFF
CTP11    0002900 3031323334353637383 9FFFFFFFFFFF
CTP11    0003000
CTP11    0003100     WCGM PAIRS
CTP11    0003200
CTP11    0003300 (82,00)
CTP11    0003400
CTP11    0003500      GRAPHIC NAMES MAX 12
CTP11    0003600
CTP11    0003700 END
```

Example 2:

```
FBSTD6   0000100 DEVICE=1403
FBSTD6   0000200 FORMAT=STANDARD,6
FBSTD6   0000300 END
FBSTD6   0000400 DEVICE=3211
FBSTD6   0000500 FORMAT=1(6,1),62(6,12),66(6,9)
FBSTD6   0000600 END
FBSTD6   0001300 DEVICE=3800
FBSTD6   0001400 FORMAT=1(6,1),62(6,12),66(6,9)
FBSTD6   0001500 END
```

The "DEVICE=" keyword signals the start of the device dependent information. For the 1403 the operator will be requested to mount the carriage tape 'STANDARD' and set the printer density to 6 lines per inch (LPI). For the 3211 and 3800 an FCB image setting the density to 6 with channel code 1 at line 1, channel code 12 at line 62, and channel code 9 at line 66 will be built.

Example 3:

```
STPAPER  0000200 PAPER=1PLY
STPAPER  0000300 FORMAT=STD6
STPAPER  0000400 CHARS=P11,H11
```

The default region of the SYSUCS data set is used by both the SETUR process and the print command. SETUR uses this region to backfill defaulted values in the SETUP request. The print command uses it to fill in defaulted values in the batch work queue. This information is used by the batch monitor to schedule print jobs on the correct printer. At print request time the 'CHARS=' keyword indicates that either a P11 or H11 train image can satisfy the print request. At SETUR time the 'CHARS=' keyword indicates that the P11 train image should be loaded in the printer. PAPER type is 1PLY regardless of printer type. The region FBSTD6 will be used to fulfill the FCB requirements based upon device type.

Example 4:

In the example that follows two picture images have been defined. Both pictures will have a pitch value of 10 as indicated by the 'PITCH=' keyword. The keyword 'CODE=' defines the displacement into the translate table where the graphic modification is to be placed. A maximum of 24 picture images may be specified in a graphic modification region. The first line of each picture image must specify the code and pitch value. The second line, in the above example, is optional and is used for reference purposes only. Each picture image must have 24 lines. The system will accept a maximum of 18 characters per line. Short lines will be padded to the right with blanks, long lines will be truncated.

```
GFGRF1   0000100 CODE=5B  PITCH 10
GFGRF1   0000200 1234567890123456789
GFGRF1   0000300
GFGRF1   0000400
GFGRF1   0000500
GFGRF1   0000600
GFGRF1   0000700     ***      ***
GFGRF1   0000800    *****    *****
GFGRF1   0000900     ***      ***
GFGRF1   0001000
GFGRF1   0001100    ***        ***
GFGRF1   0001200    ***        ***
GFGRF1   0001300    ***        ***
GFGRF1   0001400    ***        ***
GFGRF1   0001500    ***        ***
GFGRF1   0001600    ***        ***
GFGRF1   0001700    ***        ***
GFGRF1   0001800    ***        ***
GFGRF1   0001900    ****      ****
GFGRF1   0002000     ***********
GFGRF1   0002100      *********
GFGRF1   0002200
GFGRF1   0002300
GFGRF1   0002400
GFGRF1   0002500
GFGRF1   0002600
GFGRF1   0002700 CODE=7B  PITCH 10
GFGRF1   0002800 1234567890123456789
GFGRF1   0002900
GFGRF1   0003000
GFGRF1   0003100
GFGRF1   0003200
GFGRF1   0003300     ***      ***
GFGRF1   0003400    *****    *****
GFGRF1   0003500     ***      ***
GFGRF1   0003600         ***
GFGRF1   0003700         ***
GFGRF1   0003800        *****
GFGRF1   0003900       *** ***
GFGRF1   0004000       *** ***
GFGRF1   0004100       *** ***
GFGRF1   0004200      ***   ***
GFGRF1   0004300      ***   ***
GFGRF1   0004400     ***********
GFGRF1   0004500     ***********
GFGRF1   0004600    ***        ***
GFGRF1   0004700    ***        ***
GFGRF1   0004800
GFGRF1   0004900
GFGRF1   0005000
```

SETVLOCK -- Set VM Lock (0)

SETVLOCK is used to set a VM Lock.

Name	Operation	Operand
[symbol]	SETVLOCK	lock,log [,SET=set]

lock
specifies the VM Lock to be set.

Specified as: an RX address.

log

specifies the VM Lock Anchor to be used to record the status of the specified lock.

Specified as: the symbol naming a LOGVLOCK macro.

set

specifies an address in the current module to be branched to if the specified lock is already marked "set".

Specified as: an RX address.

Default: The status of the lock will not be checked.

Execution: If the branch address is specified and if the VM Lock Anchor indicates "set" the branch will be performed. Otherwise, the specified VM Lock will be set and the VM Lock Count (ISAVLKCT) in the task's Interrupt Storage Area (CHAISA) WILL BE INCREMENTED. The address of the lock will be saved in the VM Lock Anchor for use by OPNVLOCK, etc.

CAUTION: This macro must be protected from task interrupts by ITI/PTI.

Programming Note: Refer to VM Locking in Section 3.

SETXP — Set External Page Table Entries (R)

The SETXP macro instruction allows a range of virtual storage to be associated with a set of external storage addresses. It also flags pages as "unprocessed by dynamic loader." The first reference to the page or pages will then cause control to be given to the dynamic loader.

Name	Operation	Operand
[symbol]	SETXP	

Note: There are no operands.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding SETXP. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding SETXP must be issued with the PRIVILEGED option.

Execution: The first bit of the halfword immediately following the SVC is interpreted as a flag. If this bit is 1, the high-order bit of the SDA indicates which entry has been processed by the loader. The maximum page count is 1022. The low-order 10 bits of the halfword following the SVC are interpreted as a page count. The first fullword following the SVC contains the virtual storage address at which the external page table entries are to be set. After this word -- and depending on the page count -- are a number of words; each word contains an external storage address that is to be associated with a page in the virtual storage range. If the unprocessed-by-loader flag is set for a page, the first reference to that page by a program causes control to be given to the dynamic loader via a task-program interruption type 16.

The external page table entries supplied in the parameter list are set as indicated. The unprocessed-by-loader bit is set for each page whose bit string flag is a 1 and the high-order bit of the SDA is zero. This allows a mixed list to be processed.

Return Data: None.

Example: Suppose that you want to set external page table entries for three pages beginning at location NEW. You might write:

```
SAMPL   EX      0,SET
        B       SOMEPLACE
SET     DS      0F          SVC MUST BE ON FULL WORD BOUNDARY
        SETXP
        DC      H'3'        NO BIT STRING, THREE PAGES
        DC      A(NEW)      ADD EXTERNAL PAGE TABLE ENT AT NEW
        DC      H'12'       SYMBOLIC DEVICE NUMBER
        DC      H'115'      RELATIVE PAGE ON DEVICE
        DC      H'35'       SYMBOLIC DEVICE NUMBER
        DC      H'51'       RELATIVE PAGE ON DEVICE
        DC      H'12'       SYMBOLIC DEVICE NUMBER
        DC      H'34'       RELATIVE PAGE ON DEVICE
```

SETXTS -- Set Up Extended Task Status Index Field (R)

The SETXTS macro instruction enables you to set the estimated run time of your task in the XTSI.

Name	Operation	Operand
[symbol]	SETXTS	[field]

field
specifies the XTSI field to be set.

Specified as: ESTIM, which indicates that the estimated run-time field of the XTSI is to be set; SET24, which indicates 24-bit machine addressing is to be used; or, if the decimal value of 3 (for ESTIM) or 12 (SET24) is first loaded into register 15, as: (15).

Default: It is assumed that the issued has placed a value in register 15.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding SETXTS. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding SETXTS must be issued with the PRIVILEGED option.

Execution: The value in registers 0 and 1 when SETXTS is issued is stored in the extended task status index field indicated by the code in register 15.

Example: Suppose you want to set the estimated run-time field of the extended task status index. You could write:

```
         SR       0,0
         L        1,=F'runtime'
NAME     SETXTS   ESTIM
```

SIPEHOOK -- System Performance Evaluation (O)

The SIPEHOOK macro instruction is assembled into various resident supervisor modules so that system data may be collected by the System Internal Performance Evaluation Module (SIPE)

Name	Operation	Operand
[symbol	SIPEHOOK	number-value,hookcode-value

number
specifies a unique number for this SIPEHOOK within this assembly module.

Specified as: a two digit decimal number.

hookcode
specifies which SIPE collector is to be activated because of this hook.

Specified as: a three digit decimal number.

Execution: The action that occurs when a hook is reached is actually determined by the setting of an instruction switch located in the prefixed storage area (PSA) of main storage. (PSA is the term used to describe main storage locations 0-4095, which can be addressed without a base register.). When TSS startup is completed, this instruction switch contains a NOPR instruction (actually, a two-byte BCR isntruction, with condition code 0).

When control arrives at a hook, this central switch is the subject of an EXECUTE instruction. If SIPE is not being used, the NOPR instruction is executed, and control flows through the hook. However, if SIPE is active, the initialization phase of SIPE has reset this central switch to an SVC. This SVC is executed by the hook and results in a transfer of control to SIPE, which recognizes the SVC code as denoting hook execution. Basically, the following events occur for a selected hook:

1. The hook is entered, executing the switch in the PSA region (SVC).

2. The hardware-stored SVC old PSW contains the current machine status and the instruction counter.

3. The SVC new PSW becomes active.

   (a) SIPE saves all machine registers.

   (b) SIPE locates the hook via the SVC old PSW (instruction counter) and inspects the hook identity code (a constant included in the hook).

   (c) A collector is given control to abstract the appropriate data for this hook and file it in the output buffer.

   (d) The I/O buffer is output if necessary.

5. The machine registers are restored.

6. The SVC old PSW is loaded, returning control to the host module at a point just past the hook.

Example: Suppose SIPE collector 145 is to be activated in a supervisor module. The macro instruction might be written:

```
SIPEHOOK  01,145
```

This would generate:

```
EX  0,PSASIP
NOP *-*
ORG *-2
DC  AL1(145)
DC AL1(255)
```

STORE -- Store Register Contents (O)

The STORE macro instruction stores the contents of one or more registers.

Name	Operation	Operand
[symbol]	STORE	area,(first register[,last register])

area
: specifies the address of the storage area in which the register or registers are to be saved.

  Specified as: An RX address, or register notation. If register notation is used, the address must first be loaded into the specified register.

first register
specifies the first in a range of registers whose contents are to be saved, or the only register whose contents are to be saved.

Specified as: A decimal number from 8 through 15.

last register
specifies the last register in a range of registers.

Specified as: A decimal number not greater than 15.

Default: Only the register specified in the first register operand is saved.

Programming Notes: The area must be large enough to contain the specified range of registers.

STXTR -- SET and XTRCT Table

The STXTR is a macro used for generating internal tables for use by the three SET/XTRCT routines -- CEAH2, CEAS2, and CEAS4.

Name	Operation	Operand
[symbol]	STXTR	table,field,type

table
specifies the name of the dsect which is used in each particular routine.

Specified as: CHATSI for CEAH2
CHASYS for CEAS2
CHAXTS for CEAS4

field
specifies the field within the dsect which is to be SETup or XTRCTed.

Specified as: any field within the particular dsect used.

type
specifies whether the field can only be XTRCTed or also SETup.

Specified as: SETUP - setup or extracted
XTRCT - extracted only

Programming Notes: The table generated is in a standard form that the SET/XTRCT modules interpret to perform the correct movement of data from virtual memory to the corresponding supervisor tables.

SYSER -- Indicate Nonresident-Program-Detected Error (0)

The SYSER macro instruction is the means by which a nonresident program reports errors it has detected.

Name	Operation	Operand
[symbol]	SYSER	error type,fillin,id1,id2,id3,call

error type
specifies the type of error detected.

Specified as: One of the codes shown in Figure 23 under the ERROR macro instruction.

fillin
must be included for compatibility.

Specified as: Any two-digit decimal number in the range 00 through 27.

id1
is the first of three unique identifiers for the message to be issued when SYSER is invoked.

Specified as: A decimal number in the range 1 through 83.

id2
the second of three unique identifiers for the message to be issued when SYSER is invoked.

Specified as: A decimal number in the range 1 through 99.

id3
the third of three unique identifiers for the message to be issued when SYSER is invoked.

Specified as: A decimal number in the range 1 through 999.

call
is used to identify one of several calls in a module.

Specified as: A decimal number from 1 through 99.

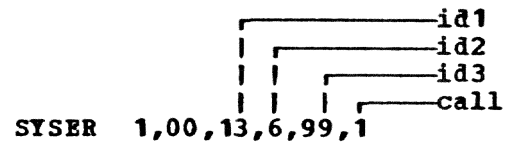
Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding SYSER. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding SYSER must be issued with the PRIVILEGED option.

Execution: The processing unit receiving the SYSER SVC stops all other processing units in the system. A message (see "SYSER DUMP" in Section 5) is issued at the operator's terminal, the system enters the wait state, and, at the installation's discretion, a dump is taken.

If the error type is 2 (major software), a program interruption 202 is queued on the calling task; this ultimately results in its abnormal termination. If the error type is 3 (hardware failure), the SVC 228 routine transfers control to the recovery nucleus. If the error type is 1 (minor software), or if the recovery nucleus returns control to the SVC 228 routine, all other processing units in the system are restarted; control is then returned to the instruction following the SYSER parameter list.

Programming Note: Part of the message issued at the operator's terminal is a nine-digit SYSER code; this code is formatted from the id1 (aa), id2 (bb), id3 (ccc), and call (nn) operands of the SYSER macro instruction and has the form aabbcccnn. This construction permits you to identify calls to the system error processor from privileged virtual storage modules to facilitate debugging. You might, for example, assign a particular id1 code to a group of related modules, assign a particular id2 code to a subset of this group, and a particular id3 code to a module or group of modules within this subset; such an arrangement would identify the source of the call to the system error processor. You could then, using the call operand, assign sequential numbers to the SYSER calls is-

sued by that module or group of molules to aid recognition of particular errors resulting in calls within the sequence. For example, you might write:



and the resulting SYSER code, 130609901, would identify the error which resulted in the call to the system error processor.

To avoid the possibility of issuing different SYSER calls with the same SYSER code (thus duplicating the messages issued at the operator's terminal and creating confusion as to the reason for the call), see System Messages for those SYSER codes already in use in the system.

Example: Suppose your task detects a minor software error and you want to get just the basic SYSER output. You might write:

```
BUG   SYSER    1,00,2,0,23,01
```

TSEND -- Force Time Slice End (R)

The TSEND macro instruction forces on your task an early time slice end.

Name	Operation	Operand
[symbol]	TSEND	

Note: There are no operands.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding TSEND. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding TSEND must be issued with the PRIVILEGED option.

Execution: The current time slice of the task issuing the SVC is terminated.

Example: If you want to cause your current time slice to come to an end, you might write:

```
XYZ      TSEND
```

TSTVLOCK -- Test VM Lock (O)

The TSTVLOCK macro is used to test the recorded status of a VM lock.

Name	Operation	Operand
[symbol]	TSTVLOCK	log,[set],[open]

log
specifies the VM Lock to be tested.

Specified as: the symbol naming a LOGVLOCK macro.

set,open
specify addresses in the current module to be branched to if the lock is marked "set" or "open", respectively.

Specified as: RX addresses.

Execution: The specified VM Lock Anchor is tested, and the appropriate branch is executed.

Programming: Refer to VM Locking in Section 3.

TWAIT -- Wait for Terminal I/O Interruption (R)

The TWAIT macro instruction checks for a response to a message you have sent and, pending its arrival, puts your task in the delay state, which causes your task's pages to be moved to auxiliary storage.

Name	Operation	Operand
[symbol]	TWAIT	

Note: There are no operands.

Execution: The SVC must be the subject of an Execute instruction and must occupy the second halfword of a fullword control block called an event control block (ECB). The resident supervisor checks the second bit of the halfword preceding the supervisor call and interprets this bit as the event complete bit. If this bit is 1, the supervisor returns control and the SVC is in effect a NOP (no operation). If the bit is 0, and there are any unmasked interruptions queued on the task, a NOP is also affected. Otherwise, the supervisor sets the TWAIT flag in the task's TSI to 1 and puts the task in the delay state; this causes time slice end to occur for the task. The task is removed from the delay state when any task-interruption -- if the task is enabled -- occurs.

Example: Suppose you send a message to some terminal and are waiting for a response. The posting routine associated with the IOCAL (see the IOCAL macro instruction) used to transmit the message to the terminal is responsible for setting the event-complete bit of an event control block to 1. You have reached a point in your program beyond which you do not wish to continue until the IOCAL posting routine has been entered. You might write:

```
         EX      0,TEST+2
         B       IOCOMPLETE
TEST     DS      0F              ALIGN
         DC      H'0'            POSTING FLAGS
         TWAIT
```

UFLOW -- User Flow for TSS and MTT (R)

The UFLOW macro instruction is used (for example, by the FLOW command processor) to modify or obtain either the conversational task limit and the number of current TSS users, or the multiterminal task (MTT) application user limits and the number of current users for each application.

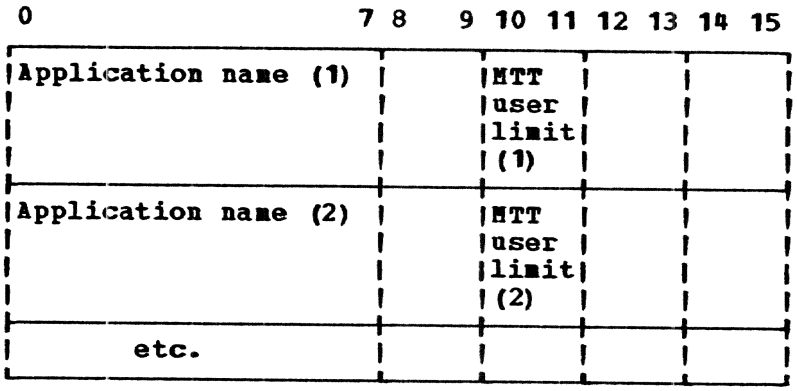
Name	Operation	Operand
[symbol]	UFLOW	

Note: There are no operands (see Initialization).

Initialization: Before executing this macro instruction, registers 1 and 0 must be loaded with the following parameters:

Register 1
An action code, specifying the action to be taken.

Code	Meaning
1	Set the conversational task limit to the value specified in the low-order two bytes of register 0.
2	Obtain the number of conversational tasks currently in execution and the conversational task limit (see Return Data below).
3	Set MTT user limits for each of the application names specified in the input buffer that is pointed to by register 0. The input buffer must contain:



The application name must be left-aligned and consist of up to eight alphameric characters; the first of which must be alphabetic. Setting the application name to X'FF' indicates the end of the buffer list, for both action codes 3 and 4. The MTT user limit must be specified in binary. The low-order bytes that are not used must contain blanks.

4	Obtain the number of MTT users in execution for each application name that is currently active in the system. Register 0 must contain the address of a virtual storage buffer into which the application names and the statistics pertaining to each can be recorded when the requested processing is performed.

Register 0
Either the conversational task limit, representing the total number of TSS users that may be logged on concurrently, if action code 1 is specified, or a virtual storage buffer address, if action code 3 or 4 is specified.

The task limit that is specified in the two low-order bytes does not affect users that are already logged on. If set to zero, no additional users may logon to TSS; the maximum must always be less than or equal to the system maximum. Any virtual storage address that is specified should point to a buffer that starts on a page

boundary and does not exceed a page length. This address can point to either an input buffer (see action code 3) or to an output buffer (see action code 4). Two system DSECTs, CHAOFL and CHAUFN, respectively, are available within TSS for use in referring to fields in those buffers.

A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to coding UFLOW. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued, prior to coding UFLOW, must be issued with the PRIVILEGED option. To ensure (for action codes 3 and 4) that the buffer will be in main storage when UFLOW is executed, an MVC instruction must immediately precede the UFLOW macro instruction.

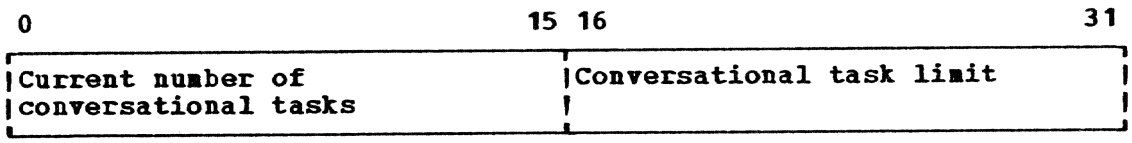
CAUTIONS: Use of UFLOW (which produces an SVC 187) is restricted to tasks having system programmer authority (O or P). Any virtual storage buffer that is provided must not go over a page boundary.

Execution: The privilege specified by the DCLASS macro instruction is verified. If acceptable, the action code is validated. For action codes 1 and 2, the appropriate limit field, conversational TSS task limit (action code 1), or the MTT application user limit (action code 3) is set in the multiterminal status control block (MTSCB). For action codes 3 and 4, the requested statistics (current number of conversational TSS tasks or current number of MTT users on a specified MTT application) and the system maximums for such limits (see Programming Notes) are recorded in the buffer. All error conditions are identified by return codes or, for action code 3, in the original input buffer (see Return Data below).

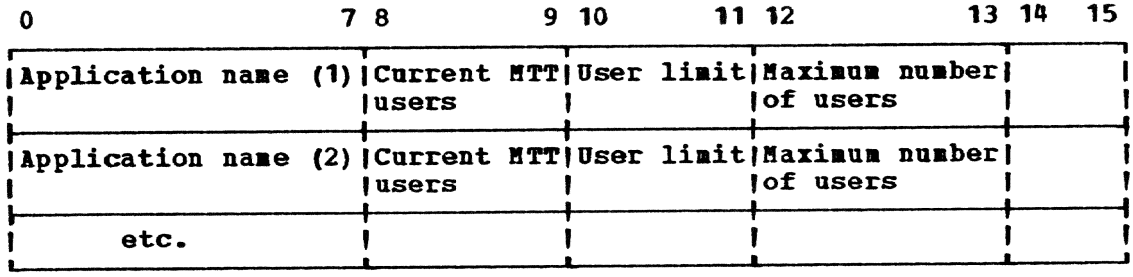
Return Data:

Register 0
For successful execution of action codes 2 and 4, contains the requested TSS or MTT statistics in the form:

For action code 2:



For action code 4:



The application name must be left-aligned and must consist of up to eight alphameric characters, the first of which must be alphabetic. When the application name field contains hexadecimal Fs, it indicates the end of the output buffer list. The low-order bytes that are not used contain blanks. The current MTT user value, the MTT user limit, and the maximum number of MTT users are all binary values.

For action code 3: register 0 points to the input buffer and may contain error indications:

1. If an application name is nonexistent, the two halfwords starting at byte 10 in the input buffer are set to X'FFFF'.

2. If the maximum allowable user limit (recorded in MTSMAX in the MTSCB) is exceeded, the two halfwords starting at byte 10 in the input buffer are set to C'**', and the maximum value is placed in the next halfword (at byte 14).

Register 15
Contains a return code:

Code	Meaning
0	Normal completion
4	Conversational task limit is larger than maximum value (MTSMAX) for action code 1
8	Action code specification error
12	Buffer exceeded page boundary

Programming Notes: The initial TSS conversational task limit is established during system generation with the TSKLMT macro instruction (see System Generation and Maintenance); the number of MTT administrators (or MTT tasks) is included in the count of conversational tasks.

The user limit specified for each MTT application program with UFLOW can never exceed the maximum value originally established by the MTT administrator when he issued an MTT command.

Before UFLOW is first issued, a GETMAIN macro instruction can be issued to get the buffer, which can be retained for the duration of the task.

If a conversational TSS task ends abnormally (completion code 2), a new task is created regardless of the conversational limit.

A command, FLOW, available only to system managers, administrators (see Managers and Administrator's Guide), and operators (see Operator's Guide), can be used dynamically to modify the number of conversational or batch tasks.

UPDTUSER -- Update User Tables (O)

The UPDTUSER macro instruction causes the data pertaining to external storage that is currently in each user table in the SYSUSE data set to be updated with information from the various user catalogs and DSCBs.

Name	Operation	Operand
[symbol]	UPDTUSER	[mode]

mode
specifies whether all or select user entries in the SYSUSE data set are to be updated. Select users are those users with currently active tasks and those users owning shared data sets that are currently being accessed.

Specified as: A - all
S - select

Default: A

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a CSECT prior to issuing UPDTUSER. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding UPDTUSER must be issued with the PRIVILEGED option.

Execution: UPDTUSER updates the cumulative page count fields in the user table data set (SYSUSE) by extracting the information from each user's catalog and each referenced data set's format-E DSCB. Temporary public data sets are erased, the total number of pages assigned to each user table is changed to reflect the values indicated by their DSCBs, and the temporary and external storage allocation fields are updated.

If mode S is specified, only those entries whose users were active at the time of issuing UPDTUSER (or, if the system failed and was restarted, users who were active at the time of system failure) are updated. ("Active" here means active task or with a shared data set that was being read. The flag USEADC in the user entry indicates an active user.)

Return Data: A message signifying the completion of the update is written to SYSOUT, a return code is placed in register 15, and control is returned to the issuing program.

Return Codes	Meaning
00	Normal return
04	DSCB error or improper authority code

Example: A privileged system programmer has previously issued an RPS or CVV command, or has decided that many user tables have become obsolete.

User: UPDTUSER

System: Returns the following message to SYSOUT: "nnnn USER TABLE STORAGE ALLOCATIONS UPDATED AGAINST DSCBS."

Programming Notes: Following an RPS or CVV command, an UPDTUSER command or macro instruction should always be issued. UPDTUSER may be issued without a preceding RPS or CVV.

UPDTUSER facilitates the conversion from an old user table entry DSECT to a new one.

If the user table is suspected or known to be in error, issuing UPDTUSER causes the current catalog and DSCB information to be placed in the user table.

If the user table is up to date except for active users, which may be true following a system failure and restart, the use of mode S speeds the updating.

Any temporary public data sets are deleted by issuing UPDTUSER.

When the user table of the task issuing UPDTUSER is itself updated, the shared virtual storage of that user table is updated to correspond to the updated SYSUSE record.

USAGE -- Display Resource Usage (S)

This macro instruction obtains accounting data that has been accumulated for a user.

USAGE is described in Assembler User Macro Instructions, except for the following information that is applicable only to system programmers (authority codes O or P).

userid
: specifies the address of a location containing the userid of the user for whom the accounting data is requested. (A nonprivileged user may obtain only his own accounting data; a system programmer may obtain the accounting data of any user.) The userid at the specified location must contain one to eight alphameric characters, the first character must be alphabetic, and the userid must be delimited by X'27'.

  Specified as: A relocatable expression, or, if the address is first loaded into the specified register, in register notation.

  Default: The issuer's userid will be used.

USELOCK -- Lock User Table Entry (O)

The USELOCK macro instruction is used to lock the virtual memory copy of a user table entry.

Name	Operation	Operand
[symbol]	USELOCK	

Note: There are no operands.

Initialization: The program issuing the USELOCK macro must have previously set up base registers for task common (CHATCM) and the user table entry (CHAUSE).

Execution: The USELKCNT is loaded into general register 15 to control the number of time slice ends that will be issued. The lock byte is then tested with a TS instruction. If successful the task id is moved from task common to the user table entry and processing continues. If the TS instruction was unsuccessful, a time slice end is issued, the count in general register 15 is decremented, and the lock byte is tested again. When the count goes to zero, processing continues as if the TS instruction had been successful.

VDMER -- VAM Data Management Error Recovery (S)

The VDMER macro instruction provides an error exit for attempting recovery or issuing diagnostic messages when error conditions arise while processing VAM data sets. If used conversationally, VDMER issues diagnostic messages and returns to the user's terminal without terminating his task. If executed nonconversationally, diagnostic messages are written to the SYSOUT device, and the task is terminated.

Standard form:

Name	Operation	Operand
[symbol]	VDMER	dcb address,message id,flags

L- and E-forms:

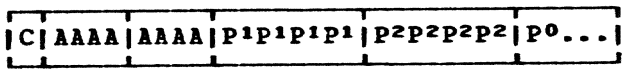
Name	Operation	Operand
[symbol]	VDMER	dcb address,message id,flags,MF={L|(E,list)}

Note: A symbol is required in the name field of the L-form. An operand omitted from the L-form must be specified in the E-form; an operand specified in the L-form is overlaid by the same operand in the E-form.

dcb address
specifies the address of the data control block (DCB) for the data set in error.

Specified as: A relocatable expression or register notation (2 through 12). If register notation is used, the address must first be loaded into the specified register.

message id
specifies the address of the second word of a parameter list that contains the identification number of the diagnostic message for the error condition. If there is variable data to be supplied for the message, pointers to the variable inserts follow the message ID, and a one-byte count of these pointers will precede the message ID (see below).



C = One-byte count of pointers (may be zero).

A = Eight-character message ID. This doubleword is addressed by word 2 of the parameter list.
$P^1,P^2,...P^0$ = Four-byte pointers to variable data, if any.

Specified as: A relocatable expression or register notation (2 through 12). If register notation is used, the address must first be loaded into the specified register.

flags
specifies the address of a two-byte field where: byte 1 indicates the type of error that occurred, and byte 2 indicates additional information about the error. The flags and their meanings are:

Byte 1	Meaning
'10'	EODAD or SYNAD condition
'20'	Clear last operation flag

Byte 2	Meaning
'0A'	Called by one of the OPEN modules CZCOA, CZCPZ, CZCOP
'0C'	SDST error in CZCOA
'0E'	Non-VAM data set in CZCOZ

Specified as: A relocatable expression or register notation (2 through 12). If register notation is used, the address must first be loaded into the specified register.

Execution: VDMER closes the data set that is causing the error. If it was open, a temporary close (CLOSE, type T) is issued on all the data sets associated with that task. Diagnostic messages are written to the

user's SYSOUT device. If conversational, control is returned to the user's task. If nonconversational, the task is terminated.

Example: A system programmer, processing a VAM data set, discovers an error condition. He issues VDMER in an attempt to recover from that condition without terminating the task associated with the data set he is processing. The three variable-data inserts to the message (that resides in SYSMLF) are automatically included in the message when it is written to the SYSOUT device.

```
R2          EQU     2
R5          EQU     5
DCBREG      EQU     3
******* PARAMETER LIST FOR VDMER **************
CNT         DC      X'03'               NUMBER OF VARIABLES FOR MESSAGE
ERRMSGID    DC      C'MSG00232'         ID OF ERROR MESSAGE IN SYSMLF
VAR1        DC      F'0'                POINTER TO VARIABLE DATA TEXT
VAR2        DC      F'0'                POINTER TO VARIABLE DATA TEXT
VAR3        DC      F'0'                POINTER TO VARIABLE DATA TEXT
            DC      AL1(L'VARBL1)
VARBL1      DC      C'TEXT'             TEXT FOR VARIABLE 1
            DC      AL1(L'VARBL2)
VARBL2      DC      C'TEXT'             TEXT FOR VARIABLE 2
            DC      AL1(L'VARBL3)
VARBL3      DC      C'TEXT'             TEXT FOR VARIABLE 3
******* END OF PARAMETER LIST *****************
VDMERFLG    DC      X'20'               CLEAR LAST OPERATION FLAG
            DC      X'0E'               NON-VAM DATA SET IN CZCOA
            .
            .
            .
            LA      6,VARBL1
            ST      6,VAR1              SET UP POINTER TO 1ST VARIABLE
            LA      6,VARBL2
            ST      6,VAR2              SET UP POINTER TO 2ND VARIABLE
            LA      6,VARBL3
            ST      6,VAR3              SET UP POINTER TO 3RD VARIABLE
            .
            .
            .
            LH      R5,VDMERFLG         SET ERROR FLAG FOR REG. NOTATION
            L       R2,ERRMSGID         SET UP MESSAGE ID FOR DIAGNOSTIC
            L       DCBREG,DCBADR       GET DCB ADDRESS
            VDMER   (DCBREG),(R2),(R5)  ATTEMPT ERROR RECOVERY
```

VSENDR -- Send Message to Task and Await Response (O)

The VSENDR macro instruction is used to send a message to another task and wait for a response from the receiving task.

Standard form:

Name	Operation	Operand
[symbol]	VSENDR	message text,reply address,reply length, message code,sender

L-form:

Name	Operation	Operand
symbol	VSENDR	message text,[reply address],[reply length], [message code],[sender],MF=L

Note: A symbol is required in the name field.

E-form:

Name	Operation	Operand
[symbol]	VSENDR	,[reply address],[reply length],[message code], [sender],MF=(E, list)

Note: An operand omitted in the L-form (except the message text) must be specified in the E-form; an operand specified in the E-form overlays the same operand that was specified in the L-form. The message text must be specified in the L-form and omitted in the E-form.

message text
specifies the text of the message to be sent.

Specified as: Text (a character string enclosed in apostrophes).

reply address
specifies the location where the message reply is to be placed.

Specified as: In the standard and L-form, a relocatable expression; in the standard and E-form, register notation; also, in the E-form only, an RX address.

reply length
specifies the length in bytes of the reply.

Specified as: In the standard and E-form, an absolute expression or register notation; in the L-form, only in register notation.

message code
specifies the message code.

Specified as: Same as for the reply length operand.

sender
specifies the ID of the task the issued VSENDR.

Specified as: Same as for the reply address operand.

Initialization: A DCLASS macro instruction with the PRIVILEGED option must be coded in a module prior to coding VSENDR. If more than one DCLASS macro instruction is issued in a module, the last DCLASS issued prior to coding VSENDR must be issued with the PRIVILEGED option.

Execution: VSENDR sets up control blocks described by DSECTs CHAMEB and CHAMCB, sets up a message flag in CHAMCB indicating the VSENDR issuer, issues the VSEND to the receiving task, and then issues an AWAIT macro instruction. The receiving task sets up a reply in the message control block and issues VSENDR.

XTRCT -- Extract Task Status Index Field (R)

The XTRCT macro instruction permits you to extract and examine one of a selected number of fields from your TSI.

Name	Operation	Operand
[symbol]	XTRCT	[tsi field]

tsi field
designates the TSI field you want to extract and examine.

Specified as: One of the codes described below, or, if a value corresponding to a desired code is first placed in register 15, as (15).

Code	Value	Meaning
USERID	1	the user ID field
SYSIN	3	symbolic device address for the input data set
SYSOUT	4	symbolic device address for the output data set
SOPRIV	6	operator/(combined with privilege class-E) system programmer privilege
SPPRIV	7	system programmer, nonprivileged
SRPRIV	8	virtual system service routines
UPRIV	9	user
CONV	10	the conversational task flag
TASKID	11	the task ID field
ITMFLG	12	the intertask message flag
XPR	13	the external priority flag
AUTH	14	the privilege field
MAV	16	the auxiliary storage requirement field
DISK	17	the auxiliary storage count field

Default: It is assumed the issuer has placed a code value in register 15.

Execution: The task status index field indicated by the code is extracted and returned to the program issuing the XTRCT.

Return Data: The extracted field is returned right-aligned in registers 0 and 1. The number of bytes returned is:

Register 15	TSI Field	Implied Length (bytes)
1	TSIUID	8
3	TSISIN	2
4	TSISOT	2
6	TSIOP (TSIF4)	1
7	TSIPP (TSIF4)	1
8	TSISP (TSIF4)	1
9	TSIUP (TSIF4)	1
10	TSICV (TSIF2)	1
11	TSITID	2
12	TSIMB (TSIF4)	1
13	TSIXPR	2
14	TSIF4	1
16	TSIARF	2
17	TSIAAF	2

CAUTION: The smallest field extracted is one byte. If you are interested in a particular bit within a byte, you must mask out the remaining bits.

Example: Suppose you want to find out if your task is being run in the conversational mode; you might write:

```
EXAMP          XTRCT              CONV
```

XTRCTL - Extract Control Registers (R)

The XTRCTL macro instruction allows you to extract and examine selected task control registers.

Name	Operation	Operand
[symbol]	XTRCTL	[field-{MCMASK|PERMASK|PERGR|PERADDR|(15)}]

field
designates the control register function you wish to extract and examine and may be written:

MCMASK - extract monitor call mask CR 8 bits 16-31
PERMASK - extract program event recording mask CR 9 bits 0-3.
PERGR - extract PER general register mask CR 9 bits 16-31.
PERADDR - extract PER address range CR 10-11.

If you choose to write register notation, you must select the proper value from the list below and place it in register 15 before issuing the macro instruction.

Code	Value
MCMASK	1
PERMASK	2
PERGR	3
PERADDR	4

Initialization: None. XTRCTL CAN BE ISSUED FROM PRIVILEGED or NONPRIVILEGED code.

Execution: The contents of selected control register save areas in the XTSI are extracted and placed in register 0 and 1. The number of bytes and the specific control register funtction to be extracted is determined by the code contained in register 15.

Code	Implied Length (bytes)
1	2
2	1
3	2
4	8

Regardless of the length, the bytes are always to be right-justified in registers 0 and 1.

Example: Suppose you wish to examine the monitor call mask in control register 8. You might write:

```
EXAMMC    XTRCTL    MCMASK
```

XTRSYS -- Extract System Table Field (R)

The XTRSYS macro instruction enables you to extract and examine one of a selected set of system table fields.

Name	Operation	Operand
[symbol]	XTRSYS	[system table field]

system table field
designates the system table field you want to extract and examine.

Specified as: One of the following codes, or, if the value corresponding to the code is first loaded into register 15, as (15).

Code	Value	Meaning
TASKINIT	3	the task initiation status field
AVAUX	5	the available auxiliary count field

Default: It is assumed that the issuer has placed a value in register 15.

Execution: The size of the field extracted is determined by the code in register 15. Corresponding lengths are shown below.

Code	Implied Length (bytes)
3	1
5	4

Return Data: The bytes in the requested field are returned right-aligned in registers 0 and 1.

Example: Suppose you want to learn the time of day that the system was IPLed. You might write:

```
NAME   XTRSYS    TOD
```

XTRXTS -- Extract Extended Task Status Index Field (E)

The XTRXTS macro instruction enables you to extract and examine one of a selected set of XTSI fields.

Name	Operation	Operand
[symbol]	XTRXTS	[XTSI field]

XTSI field
designates the XTSI field you want to extract and examine.

Specified as: One of the following codes, or, if the value corresponding to the code is fiest placed in register 15, as (15).

Code	Value	Meaning
UTIME	1	the user time field
ATIME	2	the accumulated time field
ESTIM	3	the estimated run time field
TWAIT	4	the number of TWAITs field
AWAIT	5	the number of AWAITs field
TSLICE	6	the number of time slices field
AUX-IN	7	the number of page-ins from auxiliary storage field
EXT-IN	8	the number of page-ins from external storage field
AUX-OUT	9	the number of page-outs to auxiliary storage field

EXT-OUT	10	the number of page-outs to external storage field
MDISK	11	the maximum pages used on auxiliary disk field
SET24	12	24-bit addressing flag

Default: If is assumed that the issuer has placed a value in register 15.

Execution: Registers 0 and 1 are loaded with a doubleword extracted from the issuing task's extended task status index (XTSI). The number of bytes extracted depends on the field. The field lengths for each code that can be specified are:

Code	Implied Length (bytes)
1	8
2	8
3	8
4	2
5	4
6	4
7	4
8	4
9	4
10	4
11	2

Return Data: The data returned will be right-aligned in register 1 with a left-fill (padding) of zeros.

Example: Suppose you want to find out how much time your task has used since LOGON; you might write:

```
NAME   XTRXTS   ATIME
```

ZEROSST -- Zero Statistical Recording Fields (0)

The ZEROSST macro instruction sets the statistical entries in the system to zero, thereby reinitializing the table for recording of up-to-date statistics.

Name	Operation	Operand
[symbol]	ZEROSST	

Note: There are no operands.

CAUTION: Use of SVC 194, generated by ZEROSST, is restricted to tasks having system programmer authority (0 or P).

Execution: SVC 194, which ZEROSST produces, sets the statistical recording fields in the resident supervisor to zero. These fields are: the system status table (CHASST); the schedule table counters; the page stealing counters; the scheduling counters; the drum to disk migration counters; the queue processor time and use counters.

In addition, the current time of day clock is recorded in SSTZET. This indicates to any data reduction or analysis program the zero time for the statistical recording fields.

Example: A system programmer has sampled system statistics and processed that data to evaluate the interactive user-system performance. After evaluating the data, he wants to reinitialize the statistical entries in main storage. A later sampling of statistics would show the system's performance from the time ZEROSST is executed to the next time SAMPLE is executed:

```
.
.
.
EX      0,RECORD        EXECUTES SAMPLE MACRO INSTRUCTION
.
.                       DATA REDUCTION
.
ZEROSST                 ZEROS STATISTICAL RECORDING FIELDS
.
.
.
```

PART III: SYSTEM PROGRAMMER COMMANDS

This section describes the commands and special options available only to system programmers. These commands and options supplement those available to all TSS users, described in Command System User's Guide.

In general, the commands described in this section enable system programmers to update message lists, to better control data on auxiliary storage, and perform other types of system programmer functions.

COMMAND (AND SPECIAL OPTION) DESCRIPTIONS

The general explanation of commands, the rules for entering them positionally, and the types of messages issued by each are contained in Command System User's Guide, GC28-2001.

ADDPOOL Command

This command adds a public storage pool to the system.

Operation	Operands
ADDPOOL	POOLID=pool identification [,DEVICE={2305|3330|333B|3350}] [,VOLUME=volume serial number] [,MAINT={N|Y}]

POOLID
specifies the pool identification of the public storage pool to be added to the system.

Specified as: from one to eight alphameric characters.

DEVICE
specifies the device type that the first volume in the pool is to be mounted on.

Specified as: 2305, 3330, 333B, or 3350.

VOLUME
specifies the volume serial number of the first volume in the pool.

Specified as: from one to six alphameric characters.

MAINT
specifies whether or not the pool is to be placed in maintenance status upon completion of the ADDPOOL function.

Specified as: N or Y

Default: N

Functional Description: the ADDPOOL command requires that all volumes, that make up the public storage pool, have been previously mounted and VARYed online. ADDPOOL will read the pool descriptor dataset SYSVOL, determine that all volumes are online and update the necessary tables to activate the pool. If MAINT=Y has been specified, the pool is placed in maintenance status to allow only the pre-joined user to LOGON. A pool

that is in maintenance status may only be used by the pre-joined user. To all other users, including those who share datasets in the pool, the pool looks offline. To remove a pool from maintenance status, the ADDPOOL command may be issued with just the POOLID and MAINT operands. A pool that is in maintenance status may be deleted.

Programming Notes: this command is available only to SYSOPER0, TASKID=X'0001'.

BLDPOOL Command

This command provides four functions for pool management:

1. build a new one volume pool from an empty private volume,
2. prepare an empty private volume for addition to an existing pool,
3. partition an existing pool into two pools,
4. modify a pool's descriptor dataset.

Operation	Operands
BLDPOOL	MODE={NEW|ADD|PART|MOD} ,POOLID=pool identification [,CATVOL=preferred SYSCAT volume serial number] [,MULTIVOL={N|Y}] [,DEVICE={2305|3330|333B|3350}] [,VOLUME=volume serial number] [,USERID=pre-joined user identification] [,RVN=first relative volume number of partitioned pool]

MODE
specifies the type of processing to be performed. NEW builds a new one volume pool, ADD will prepare a volume for addition to an existing pool, PART will partition a pool into two pools, and MOD will modify the pool's descriptor dataset, SYSVOL.

Specified as: NEW, ADD, PART or MOD.

POOLID
specifies the pool identification to be assigned to the new pool. This operand is required for MODE=NEW or PART.

Specified as: from one to eight alphameric characters.

CATVOL
specifies the volume serial number that will contain the SYSCAT dataset. This operand may only be specified for MODE=MOD.

Specified as: from one to six alphameric characters or one to six blanks to indicate that no volume is preferred for SYSCAT.

MULTIVOL
specifies whether or not multi-volume datasets are to be allowed within this pool. This operand may only be specified for MODE=MOD.

Specified as: N or Y

Default: N

DEVICE
specifies the device type that the new volume is to be mounted on. This operand is required for MODE=NEW or ADD.

Specified as: 2305, 3330, 333B, or 3350.

VOLUME
specifies the volume serial number of the new volume. This operand is requiredd for MODE=NEW or ADD.

Specified as: from one to six alphameric characters.

USERID
identifies a pool's pre-joined user. This operand is required for MODE=NEW or PART.

Specified as: from one to eight alphameric characters.

RVN
specifies the relative volume number within the existing pool that is to become the first volume of the partitioned pool. This operand is required for MODE=PART.

Specified as: from one to three decimal digits.

Functional description:
MODE=NEW
The specified volume is mounted, the label is read and checked for consistency, and the descriptor dataset, SYSVOL, is built. Empty datasets for SYSUSE and SYSSVCT are built and include the identity of the pre-joined user. A USERCAT that contains entries for the constructed datasets is built. Finally, the volume label is modified to include the pool and DSCB information required by an ADDPOOL command.

MODE=ADD
The specified volume is mounted, the label is read and checked for consistency, and the pool's descriptor dataset, SYSVOL, is updated to include the new volume. Finally, the volume label of the new volume is modified to include the pool information (e.g., POOLID). The new volume will become available for storage assignment the next time this pool is 'added' to the system.

MODE=MOD
The pool descriptor dataset, SYSVOL, is updated to include the specified parameters. The new parameters will take effect the next time this pool is 'added' to the system.

MODE=PART
The specified pool is checked for maintenance status; if it is not in maintenance the command is cancelled. Each user in the existing SYSSVCT is tested for RVN limit consistency and MOVEUSER status; if any user is not in MOVEUSER status, or is found to overlap the two pools, the command is cancelled. The specified relative volume number is checked for consistency and the volume label of its volume is read and checked for consistency. The descriptor dataset SYSVOL of the new poolid is built from information in the SDAT, and SYSVOL of the existing poolid is updated. Empty datasets for SYSPLIB, SYSBWQ, and SYSCAT are constructed. The datasets SYSUSE and SYSSVCT are built and include the pre-joined user. A USERCAT that contains entries for the constructed datasets is built. The existing SYSSVCT is read to determine which USERIDs are to be included in the new poolid and both the existing and new SYSSVCT and SYSUSE datasets are updated accordingly. Finally, the volume labels of the volumes in the new poolid are modified to include the pool and DSCB information required by ADDPOOL. Before partitioning a pool, all userids in the pool must be 'compressed'; i.e., a SETRVN and a MOVEUSER with POOLID=*POOL must be issued for all userids.

Programming notes: this command with MODE=NEW is available only to TSS* **** . For all other MODEs the command is available only to the pool's

pre-joined user. The initial password for the pre-joined user is IBM. When MODE=PART, an ADDPOOL must subsequently be issued against the original poolid to remove it from maintenance status (thereby adding it to the system). Similarly, an ADDPOOL must be issued against the new poolid to add it to the system.
When MODE=ADD, the new volume will not be available until first a DELPOOL and then an ADDPOOL are issued against the poolid.

Examples:

```
BLDPOOL  MODE=NEW,POOLID=TSS002,DEVICE=3350,
         VOLUME=VOL250,USERID=SYS002

BLDPOOL  MODE=ADD,DEVICE=333B,VOLUME=VOL360

BLDPOOL  MODE=PART,POOLID=TSSSYS05,USERID=TSS05,
         RVN=5

BLDPOOL  MODE=MOD,CATVOL=VOL830,MULTIVOL=Y
```

BLDSVCT Command

This command will reconstruct the SYSSVCT dataset by searching a public storage pool for USERCAT DSCBs.

Operation	Operands
BLDSVCT	POOLID=pool identification

POOLID
identifies the public storage pool on which SYSSVCT is to be rebuilt.

Specified as: from one to eight alphameric characters.

Functional description: the volumes that make up the public storage pool are searched for the 'most recent' USERCAT datasets. SYSSVCT is rebuilt from them by building a record for each USERCAT DSCB found. All RVN limit information is lost. The volume label of the first volume in the pool is updated to point to the new SYSSVCT.

Programming note: this command is available only to TSS***** and may only be issued for pools in maintenance status.

CC (Check Catalog) Command

This command allows a privileged user to run an integrity check on the catalog and compare information in the data set descriptor for VAM data sets to corresponding information in the format E DSCB.

The catalog may be inspected for one specific user, or if desired, for all users joined to the system.

Diagnostics are produced on SYSOUT describing the location and type of any irregularities unless the command was issued conversationally and the *ALL option was requested in which case any messages are written into a data set and printed on a high-speed sprinter. Also, included in this command is the ability to display and/or patch the catalog of any user.

Operation	Operand
CC	USERID={user identification|*ALL},[DISPLAY=relative page,] [WRITE=relative page],[PRIVATE=volserno]

USERID
specifies the user identification of the user whose catalog is to be examined, displayed or written.

*ALL
will check integrity of catalog for all users joined to the system.

DISPLAY
specifies the number of the relative page within the catalog which the user wishes to examine and/or patch. Cannot be used with the *ALL option.

WRITE
specifies the number or the relative page within the catalog which the user has patched and now wishes to be written out. Cannot be used with the *ALL option.

PRIVATE
specifies the volume serial number of the pack which the user wishes to have mounted to compare catalog and DSCB information for private data sets. Cannot be used with the *ALL option.

Programming Note: The CC command will be honored only for users with authority 'O'. The CC command may be used in conversational and non-conversational tasks. The DISPLAY, WRITE and PRIVATE options must be used on an individual basis.

Check Catalog Processing. The authority of the user is first validated. The DISPLAY and WRITE parameters are checked and if both appear a diagnostic is issued. Next a DCB is opened for TSS*****.SYSCAT and a FIND is issued against the userid supplied. If the *ALL option was speci-

fied, a check is made to be sure the DISPLAY, WRITE or PRIVATE parameters parameters were not entered. If not, the user identifications are selected from the POD for SYSCAT. The userid is then passed to the FIND macro and processing continues the same as if the userid were passed in the command.

If the userid does not exist, FIND will return an error code and a diagnostic is issued.

CHECK CATALOG processing is accomplished with two passes through the catalog. The first pass checks SBLOCK structure, and if no errors are found, a second pass is made which checks information in the catalog against information in the format E DSCB.

SBLOCK Checking. A GETMAIN of three pages is done for buffers. The three buffers will now be denoted primary, alternate and extra in this discussion. A call is made to the GET page subroutine for page 0 for the primary buffer. The number of pages in the user's catalog is saved and every relative page processed is checked against this number to determine if it falls within the range of pages allocated for this user's catalog. Then a SETL and GET are done to read in the relative page into the specified buffer. The fact that this relative page now resides in this buffer is noted.

The first (next) sblock is checked for "bytes in use" being equal to zero. If "bytes in use" is equal to zero, the sblock is ignored. If a backward pointer exists and the address is on a 64 byte boundary, the sblock is also skipped because in this case the sblock is either an extended sblock or a sharing list and these are checked via a different route; i.e., through the index levels pointing to them. If a backward pointer is zero, a check is made to determine if its a backward pointer to relative page 0 sblock 00. If it is, the sblock is ignored as above. The sblock identifier is now checked 03, 04, 05, and 06 and these are skipped since data set descriptors are all checked via a different mechanism. The sblock identifier can now only be an INDEX(01) or GENERATION INDEX(02). If not, a diagnostic is issued.

The count of pointers within this index level is noted. If greater than three an extended sblock(s) will later be needed. The pointer fields are now examined individually by passing them to the POINTER subroutine. The POINTER subroutine checks that the pointer address is a valid pointer. If not, a diagnostic is issued. A check is made to see if this relative page is already present in the extra buffer; if not, a call is made to GETPAGE with the extra buffer specified. Now the backward pointer of the sblock pointed to is checked to see that it points to the proper pointer field. The qualifier of the original pointer field is checked against the qualifier in the sblock. The pointer names are matched to see if they are equal. Any discrepancy is noted with a diagnostic which indicates the locations of the problems found. All pointer fields are examined in this way. The absence of a pointer field indicates that the level has been exhausted, and this is checked with the count of pointers that were indicated. If there is a discrepancy, a diagnostic is issued. At this point, the next sequential sblock is processed. When all sblocks on the first catalog page have been processed, a GETPAGE is done on the next relative page of the catalog. When all pages have been exhausted and no errors were encountered a switch is set which indicates sblock checking has been completer. If an error was found, SYSCAT is closed and the next user's catalog is examined. If processing was for a specific userid, a return is made to the command system.

Catalog/DSCB Checking. If no errors were encountered on the first pass, check catalog now does a data check on the data set descriptors and compares catalog information to DSCB information.

The first (next) sblock is checked to see if a backward pointer exists. If a backward pointer does exist, a check is made to see if it points to a sblock boundary. If so, the next sblock is obtained. If a backward pointer does exist, and the address is not on an sblock boundary, the identifier is now checked for a 06. When a 06 identifier is present, a check is made to see if the PRIVATE option was specified. If it was, the next sblock is obtained. When the identifier is an 06 and PRIVATE was not specified, a call is made to a subroutine which will do a data check for valid flags on the 06 data set descriptor. Fields which are checked within the data set descriptors are: sharing flag; sharing privileges; data set retention and and access privileges; and data set organization. If any fields are determined invalid, a diagnostic is issued and processing continues. Next, a call is made to a subroutine which will build a fully qualified name. While building an FQN, this routine checks if the FQN exceeds 44 characters, and if so, a diagnostic is issued and processing continues.

After the FQN is built the FORMAT E DSCB is obtained by issuing a SETX. Next the FQN and the data organization information in the catalog is compared to corresponding information in the DSCB; any discrepancy is noted by a diagnostic, and the next sblock is obtained.

When the identifier is a 03, a call is made to the data check subroutine and fields are checked for valid flags as above. Next a check is made to determine if this private data set resides on a disk; if not, the next sblock is obtained. However, if the private data set is of VAM format, a call is made to the subroutine which will build the FQN. If the PRIVATE option was not specified, a message is issued stating on which pack this FQN resides.

If the PRIVATE option was specified, the volume serial entered in the command parameters is compared to the volume serial number in the data set descriptor associated with this FQN. If they are equal, a call is made to Mount Request; otherwise, the next sblock is obtained.

If the identifier is other that an 06 or 03, the sblock is ignored, and the next sblock is checked.

When all sblocks on the first catalog page have been processed, a GET page is done on the next relative page of the catalog. When all pages have been exhausted, a FREEMAIN is done on the buffer pages. The DCB for TSS*****.SYSCAT is closed and a completion message is given. Exit is to the command system, unless *ALL option was specified in which case, the above steps are repeated until all users' catalogs have been searched.

When the DISPLAY keyword is present, after parameter validation and a FIND has been issued against the user's catalog, the relative page is checked against the allowable relative pages for the user's catalog. If the page is legitimate a GET page is done into the display buffer. a message is now written out giving the virtual memory location of the display buffer. The userid for this operation is saved in the event the user wants to write this page with the WRITE option. The DSCB for TSS*****.SYSCAT is closed and a return is made to the command system.

The user is now free to use PCS and/or VSS to review and/or modify the catalog page. The user may now use the write option of the command to write out any changes externally.

When the WRITE keyword is present, after parameter processing is complete and a FIND has been issued against the catalog, the input relative page number if checked for validity. Since the user is indicating that this is a page to be written out, this relative page must reside in the display buffer. A check must be made to insure that the relative page is for the proper catalog. For this purpose, the userid must match the

userid that was saved when the DISPLAY keyword was entered. A PUTX is now done from the display buffer for that relative page. The DCB for TSS*****.SYSCAT is closed and a FREEMAIN is done on the display buffer. A completion message is given and a return made to the command system.

Example:

```
  User:    CC TSS
  System:  PROCESSING COMPLETE
           SYSTEM HAS CHECKED THE VALIDITY OF TSS's catalog

  User:    CC BOYD
  System:  INCORRECT BACKWARD POINTER. REL PAGE - SBLOCK
           LOCATION OF FORWARD POINTER 00 0798. REL PAGE -
           SBLOCK LOCATION OF BACKWARD POINTER 0 10FC0
           system has discovered an incorrect backward pointer in an
           SBLOCK with address relative page 1, SBLOCK location FC0
           system has discovered an incorrect backward pointer in an
           SBLOCK with address relative page 1, SBLOCK location FC0
           the forward pointer resides at relative page 0, location 798

  User:    CC BOYD,DISPLAY=1
           user wishes to display page 1
  System:  RELATIVE PAGE 0001 IS AT VIRTUAL MEMORY LOCATION 7F000

  User:    DISPLAY L'7FFC0'.L'7FFFF'
           user displays SBLOCK in question
  System:  (PCS or VSS displays SBLOCK)

  User:    SET L'7FFC6'=X'98'
           user modifies pointer to point to correct place
  System:  (PCS sets byte)

  User:    CC BOYD,WRITE=1
           user wishes to write corrected page into catalog
  System:  PROCESSING COMPLETE
           page has been written

  User:    CC BOYD
           user wishes to check modification
  System:  PROCESSING COMPLETE
           catalog for userid BOYD is now correct

  User:    CC *ALL
           user wants to check catalog of all users joined to the system
           System:  B033 PRINT ACCEPTED
           messages are printed on high-speed printer if *ALL option
           is entered conversationally
           CC found a total of 0000 errors; messages  printed under
           DSNAME $$$000 15
           no errors were found in catalog

  User:    CC TSS,PRIVATE=RELPAK
           user wishes to check DSCBs on private pack RELPAK
  System:  PROCESSING COMPLETE
           catalog and corresponding DSCBs on RELPAK have been checked
```

CLASSGTF Command

This command translates the captured trace data into formatted printable hexadecimal characters with labels.

Operation	Operand
CLASSGTF	INDSN=dsname1 [,OUTDSN=dsname2] [,CLASS=class[,...]] [,*B]

INDSN
the name of the trace data set to be translated.

Specified as: a 1 to 35 character name; the first character must be alphabetic.

Default: none.

OUTDSN
the name of the output data set that will contain the translated trace data.

Specified as: a 1 to 35 character name; the first character must be alphabetic.

Default: if defaulted, the translated data will go to the terminal only.

CLASS
identifies the data classes to be translated.

Specified as: MACHINE

Default: none.

*B
specifies that translated data is to go to the terminal as well as dsname2.

Specified as: B

Default: translated data will not go to the terminal.

CNVTPOOL Command

This command is used to convert an existing TSS system (pre-public storage pools) to a one pool system. The CNVTPOOL command is issued only once for each TSS system.

Operation	Operands
CNVTPOOL	POOLID=pool identification [,CATVOL=SYSCAT volume serial number] [,MULTIVOL={N|Y}]

POOLID
specifies the pool identification to be assigned to the system pool.

Specified as: from one to eight alphameric characters.

CATVOL
specifies the volume serial number of the volume to be used to contain the SYSCAT dataset.

Specified as: from one to six alphameric characters.

MULTIVOL
specifies whether or not multi-volume datasets are to be allowed within this pool.

Specified as: N or Y

Default: N

Functional description: CNVTPOOL is a one-time command used to build the necessary datasets and modify the volume lablels of an existing TSS system for public storage pools. The pool descriptor dataset, SYSVOL, is built from information in the public volume table (CHBPVT). The SYSVOL, SYSCAT, and SYSSVCT DSCB pointers are located and moved to the volume label of relative volume zero. All volume labels are modified to contain the pool identification.

Programming note: this command is available only to TSS*****. The userid TSS***** becomes the pool's pre-joined userid.

DDEF -- Define a Data Set

An explanation and the general format of the DDEF command is contained in Command System User's Guide. The operand choices available to the TSS user with privilege class E (system monitor) are described below and are in addition to the operand choices described in Command System User's Guide.

data set organization
specifies a two-character code that indicates the organization of the data set.

Specified as: MS - MSAM (multiple sequential access method)

UNIT=
specifies the type of device needed for the data set.

Specified as: PR (printer), PC (punch), RD (card reader), or a four-digit hexadecimal number (0001 to 7FFF) assigned at system generation to the desired I/O device as its symbolic device address. (See in Part I, "Reserving I/O Devices for a Nonconversational Task.")

Note: The DISP keyword operand must be OLD when using the card reader and NEW when using the card punch or the printer.

DELPOOL Command

This command deletes an active public storage pool from the system.

Operation	Operands
DELPOOL	POOLID=pool identification [,FORCE={Y|N}] [,CATFLUSH={Y|N}]

POOLID
identifies the public storage pool to be deleted from the system.

Specified as: from one to eight alphameric characters.

FORCE
specifies whether or not active users of the pool are to be forced off the system so that the pool may be deleted.

Specified as: N or Y

Default: N

CATFLUSH
specifies whether or not the catalogs are to be flushed from the SYSCAT dataset.

Specified as: Y or N

Default: Y

Functional description: DELPOOL will delete information in system control blocks and logically remove the public storage pool. If FORCE=Y, all users logged on in the pool will be 'forced', all open shared datasets in the pool will be closed and their users will ABEND with completion code 1; all JFCBs for all datasets in the pool will be released,

and the pool is deleted. If FORCE=N the pool will not be deleted if any user is logged on to the pool or if there are open shared datasets in the pool; instead, a list of USERIDs, along with their POOLID, that are logged on or that have open datasets is displayed, and the pool is marked in 'delete status' to prevent new users from logging on, and prevent additional shared datasets from being opened. After all users have logged off and all shared datases have been closed, DELPOOL must be reissued to delete the pool.

Programming note: this command is available only to SYSOPER0, TASKID=X'0001'.

DISP Command

This command will display the contents of all or a specified part of the storage of an NCP/PEP or EP.

Operation	Operands
DISP	resource [,INITLOC=] [,FINLOC=] [,DSNAME=] [,NCPDES=]

resource
The name of the resource that is to be displayed.

Specified as: a 1-8 character name, the first character must be alphabetic. This name must also be the name of a region in the TSS*****.SYSRCS dataset.

INITLOC
the initial location of storage that is to be displayed.

Specified as: a 1-8 hexadecimal number.

Default: zero.

FINLOC
the final location of storage that is to be displayed.

Specified as: a 1-8 hexadecimal number.

Default: last storage location.

DSNAME
the name of the output dataset that will contain the storage dump as specified above.

Specified as: a one to thirty-five character name. The first character must be alphabetic. Note: If this parameter is omitted, the output will go only to the terminal.

NCPDES
directs the output to the dataset only or to the dataset and the terminal.

Specified as: LD for output to the dataset only or LDT for output to both the dataset and the terminal.
Note: if the DSNAME has been defaulted the display will be directed to the terminal only regardless of this parameter.

Default: There is no default for this parameter.

Example 1: the user wants to display the storage of the EP06 resource from location X'5000' to X'10000', and he wants the data to go to both his terminal and the 'DISPOUT' dataset:

```
disp ep06,5000,10000,dispout,ldt

or

disp resource=ep06,initloc=5000,finloc=10000,dsname=dispout,
     ncpdes=ldt
```

Example 2: the user wants to display the storage or the EP06 resource from location X'5000' to X'10000',and he wants the data to go to his terminal only:

```
disp ep06,5000,10000

or

disp resource=ep06,initloc=5000,finloc=10000
```

DSCBS Command

This command searches a public storage pool for DSCBs (DSCB types are specified by the user), makes an output list of DSCB data, and optionally, catalogs uncataloged datasets.

Operation	Operands
DSCBS	[LIST={ALL|ERROR|NOTCAT|ALLERR}]
	[,CATALOG={N|Y}]
	[,CLEAN={N|Y}]
	[,VOLUME=volume serial number]
	[,STARTDS=start data set name]
	[,REPORTDS=listing data set name]
	[,USERID=user identification]
	[,CHKCAT={N|Y}]

LIST

specifies the type(s) of DSCBs to be searched for (and listed).

Specified as:

ALL - specifies that all DSCBs (including the DSCBs for noncataloged datasets and DSCBs that have errors) are to be searched and 'listed'.

ERROR - means that only DSCBs having errors are to be listed in REPORTDS; among the errors checked are checksum and DSORG.

NOTCAT - means the search and listing involves only those DSCBs that are not cataloged.

ALLERR - means the search and listing involves only those DSCBs that have some error, or are not cataloged.

CATALOG
specifies whether or not uncataloged datasets are to be cataloged (see CHKCAT).

Specified as: Y or N

Default: N

CLEAN
specifies whether or not datasets which cannot be cataloged because of name or generation conflicts are to be erased.

Specified as: Y or N

Default: N

VOLUME
specifies a single volume serial number within the pool that is to be used for the search.

Specified as: from one to six alphameric characters.

Default: the entire pool.

STARTDS
specifies the dataset name at which the search is to begin. This operand is used to restart a previously aborted search.

Specified as: from 10 to 44 character dataset name including the USERID.

Default: the search starts at the first volume of the pool.

REPORTDS
specifies the dataset name of the listing dataset. The dataset organization depends upon the installation's sysgen value, or the user's default value: if the dataset is VI, entries will be made in alphabetic order; if the dataset is VS, entries will be made in the order found on the pool volume(s).

Specified as: a one to 35 character dataset name. If not specified, the user must DDEF a dataset with a DDNAME of DSCBSOUT before executing this command.

USERID
specifies the user identification of the DSCBs to be searched for.

Specified as: from one to eight alphameric characters; the first must be alphabetic.

Default: all USERIDs.

CHKCAT
specifies whether or not the catalog is to be checked for uncataloged datasets.

Specified as: Y or N. If Y, all uncataloged datasets will be so marked in the REPORTDS dataset.

Default: Y

Functional description: the DSCBS command reads all DSCBs on the public volume(s) specified and writes a record into the REPORTDS for all DSCBs that match the input parameters. If the CATALOG option is specified, an attempt will be made to catalog all uncataloged DSCBs that are found. The REPORTDS will reflect the current status of each DSCB read. This command can be used to print the status of all DSCBs on a pool.

Programming note: this command is available only to a pool's pre-joined user, an administrator, or a manager.

DUMPRES Command

This command will cause the storage of the resource to be recorded.

Operation	Operands
DUMPRES	resource [,DSNAME=] [,DMPPH1=] [,DMPPH2=] [,DUMPDS=]

resource
identifies the resource to be dumped.

Specified as: a 1-8 character name, the first character must be alphabetic. This name must also be the name of a region in the TSS*****.SYSRCS dataset.

Default: There is no default for this parameter.

DSNAME
the name of the dataset that contains the load modules that will be used to transmit the data from the resource. These modules will be referred to as the dump routines.

Specified as: a one to thirty-five character name. The first character must be alphabetic. This is the name of the dataset that contains dmpph1 and dmpph2.

DMPPH1
the name of the load module which is the first phase of the NCP dump routine.

Specified as: a one to eight character name. The first character must be alphabetic.

Default: user specified in the TSS*****.SYSRCS dataset.

DMMPH1
the name of the load module which is the second phase of the NCP dump routine.

Specified as: a one to eight character name. The first character must be alphabetic.

Default: user specified in the TSS*****.SYSRCS dataset.

DUMPDS
the name of the dataset that will contain the record of the resource storage.

Specified as: a one to thirty-five character name. The first character must be alphabetic. This is the name of the dataset that will contain the dump output.

EVV (Enter VAM Volume) Command

This command catalogs private VAM volumes by volume.

Operation	Operand
EVV	DEVICE=device type,VOLUME=(volume serial number[,...]) [,USERID=user identification]

DEVICE=
specifies the type of direct access device that the VAM volume is on.

Specified as: 2311, 2314, 3330, 333B, or 3350.

VOLUME=
identifies the volume or volumes to be processed.

Specified as: One to six-alphameric characters composing the volume serial number for each volume; the volume serial numbers are enclosed in one set of parentheses.

USERID
specifies user identification, which is used to maintain compatibility between systems; this operand is available only to the system administrator or the system manager.

Specified as: One to eight characters, the first of which is alphabetic. (If less than 8 characters, the system will pad to the right with asterisks.)

Default: The issuer's user identification.

Functional Description: EVV catalogs all data sets on the volumes specified.

Programming Notes: EVV allows the user to introduce VAM data sets created under other TSS systems to his current installation, or to recatalog previously deleted VAM data sets. Private VAM data sets created under TSS are automatically cataloged.

Example: User AMYB1 has three private volumes which are necessary for the execution of his program; he wants to enter them into the system.

User: evv 2311,(111500,111501,111502),amyb1
System: (Makes catalog entries for all data sets residing on the specified volumes.)

FIXCAT Command

This command will correct or rebuild a catalog by checking the user's catalog datasets against public storage and cataloging any uncataloged datasets, or optionally, will completely rebuild the catalog datasets from dscb information.

Operation	Operands
FIXCAT	USERID=user identification [,SYSCAT2={N|Y}] [,SYSCAT={N|Y}] [,NOFIX={N|Y}]

USERID
identifies the user whose catalog datasets are to be fixed.

Specified as: from one to eight alphameric characters; the first must be alphabetic.

Default: none.

SYSCAT2
specifies whether or not the SYSCAT2 dataset is to be erased and replaced by the SYSCAT dataset. If this option is specified, the new SYSCAT2 dataset will then be checked for errors.

Specified as: Y or N

Default: N

SYSCAT
specifies whether or not the SYSCAT dataset is to be erased and replaced by the SYSCAT2 dataset. If this option is specified, the SYSCAT2 dataset will be checked for errors.

Specified as: Y or N

Default: N

Note: If both SYSCAT2=Y and SYSCAT=Y both datasets will be erased and rebuilt by cataloging all of this user's datasets found on public storage. Note that this will result in the loss of all sharing information and also the loss of all entries of private datasets.

NOFIX
specifies whether or not any changes are to be made to the catalog datasets. If NOFIX=Y is specified and an error is found in the catalog, a diagnostic will be issued stating that an error was found, but no changes will be made to the catalog datasets.

Specified as: Y or N

Default: N

Functional description: The SYSCAT dataset is the backup copy of the catalog and the SYSCAT2 dataset is the working copy of the catalog. The FIXCAT command will cause the SYSCAT2 dataset to be verified first. All pointers within the SYSCAT2 dataset will be checked and if an error is found, the bad entry will be deleted. When the internal checking is completed, public storage will be searched and any additional datasets will be cataloged.

If the SYSCAT2 dataset cannot be used, the SYSCAT dataset will replace the SYSCAT2 dataset and catalog verification will continue as stated above.

Only if both the SYSCAT and SYSCAT2 datasets cannot be used, will the catalog be completely rebuilt by scanning public storage.

Programming note: This command is available only to the pool owner.

FIXDSCB Command

This command allows a privileged user to analyze and optionally rebuild a DSCB chain.

Operation	Operands
FIXDSCB	DSNAME=dataset name [,USERID=user identification] [,PATSCAN={Y|N}] [,FIX={Y|N}]

DSNAME=
identifies the dataset name of the DSCB chain to analyze.

Specified as: a fully qualified dataset name

Default: none.

USERID=
specifies the dataset owner identification.

Specified as: three to eight characters, the first of which is alphabetic. (If less than eight characters, the system will pad to the right with asterisks.)

Default: the command issuer's userid.

PATSCAN=
specifies whether or not to scan the PAT for DSCB pages and add back into the DSCB chain any format "G" slots which are not in the chain.

Specified as: Y - scan PATs for lost slots
N - no PAT scan

Default: N

FIX=
specifies whether or not to correct any slot errors encountered while scanning the DSCB chain.

Specified as: Y - correct errors
N - no error correction

Default: N

Functional Description: FIXDSCB will obtain the format "E" DSCB pointer by calling LOCFQN. After reading the format "E" it compares the name in the DSCB to that in the returned catalog sblock. If the names do not match an error message is issued and processing is terminated.

FIXDSCB can only be used with type 2 format "E" DSCBs connected to FORMAT "G" DSCBS. If the format "E" is not the correct type processing is terminated.

All DSCBs in the chain are read in and error checked, and an appropriate error message is issued for any errors encountered. A table (CHATBL) is built in virtual memory describing the DSCB chain. This table contains such things as the DSCB sequence counter, address and error indicators and is used in conjunction with the PATSCAN and FIX options of the command.

If the PATSCAN option was selected, FIXDSCB, upon completion of the error analysis phase, will scan the PATs for lost DSCBS. Any format "G" DSCBS whose format "E" pointer matches that of the format "E" DSCB, and are not in the DSCB table (CHATBL) will be added to the end of the table. After the PATs are scanned lost DSCBs will be added to the end of the DSCB chain based upon the sequence counter in the DSCB slot, lowest slots added in first.

After error analysis and PATSCAN, FIXDSCB will correct slot errors and update DSCB page counts if the FIX option was selected. Slots with invalid format "E" pointers are removed from the chain, external page entry counts are corrected, sequence counters are corrected, and CHECKSUMs are computed. Finally the format "E" is rewritten with the correct page counts.

FIXVI Command

This command allows a privileged user to rebuild the directory for a broken VISAM data set or VISAM member of a partitioned data set.

Operation	Operand
FIXVI	DSNAME=data set name[(member name)] [,USERID=user identification] [,PATSCAN=Y|N]

DSNAME=
identifies the dataset, residing on direct access storage, that is to be rebuilt. The data set must be in VISAM II format.

Specified as: a fully qualified data set name and (optionally) a member name of a VPAM data set. When specified, the member name is enclosed in parentheses and immediately follows the VPAM data set name.

USERID=
specifies the user's identification of the data set owner.

Specified as: one to eight characters, the first of which is alphabetic. (If less than 8 characters, the system will pad to the right with asterisks.)

Default: the user's identification.

PATSCAN
specifies whether or not to scan the PAT pages for X'41' pages and add back into the dataset any pages whose format 'E' DSCB pointer matches the format 'E' pointer for the dataset.

Specified as: Y - scan PATs for additional pages.
N - no PAT scan.

Default: N

Functional Description: FIXVI will reconstruct the directory for a VISAM data set or a VISAM member of a partitioned data set. FIXVI should be used when a VISAM data set becomes unuseable. This is usually indicated by error messages out of the VISAM access method routines.

When invoked, FIXVI will first delete the existing directory. Next, it will save the count of data pages in the data set and set the RESTBL/ member header page count to zero. Pages are then input (via SETXP) and added back into the data set one at a time. As each page is input, it will be validity checked before being added back into the data set. The format 'E' pointer on the page is compared to the format 'E' pointer of the data set for VI data sets, and the unique member pointers on the page are compared to each other for VI members of partitioned data sets. Pages with bad format 'E' or member pointers will be deleted.

FIXVI uses SETL and ADE (add directory entry) to perform data set positioning and directory rebuilding. As each data page is input, FIXVI will locate the record with the lowest key and use this as input to the VISAM SETL routine. If the key is not a duplicate, the page is added to the data set and ADE is called to update the directory.

The PATSCAN option, when selected, will be performed after existing data pages have been processed. Only pages whose format 'E' pointers match that of the dataset are called back.

Empty pages, pages with invalid format 'E' pointers or member pointers, and pages with duplicate keys will be deleted from the data set. An appropriate message will also be sent to SYSOUT stating the reason for the deletion and the RPN of the page deleted. After all pages have been processed and the directory rebuilt, the user is prompted as to whether or not to output the new directory.

GTF (General Trace Facility) Command

The GTF command will cause the data specified (interrupts, events, etc.) to be recorded in the manner specified.

Operation	Operand
GTP	{[IO={Y|N}][,XPI={Y|N}][,SDA=sda[-sda]][,RID=rid[-rid]] [,SVC=svc[-svc]][,PI=pi[-pi]][,EXT=ext]-ext]][,MC=mc[-mc]] [,RTAM=[*ENT][,*PIU]]
	|TASK=([TASKID=taskid] [,SVC=svc[-svc]] [,PI=pi[-pi]] [,EXT=ext[-ext]] [,RTAM={Y|N}])}
	,{DSNAME=dsname|NOREPROT=Y} [,REPORT=Y] [,TRACID=number] [,END=Y]

Note: all operands are keyword.

IO
specifies whether or not I/O recording is to be done.

Specified as: Y or N

Default: N

XPI
specifies whether or lnot external program interrupts are to be included in the trace.

Specified as: Y or N

Default: N

SDA
the symbolic device address or a range of symbolic device addresses from which data is to be recorded.

Specified as: a decimal integer(s) or as a hexadecimal integer(s). A hexadecimal integer is preceded by the letter X and the integer is enclosed in single quotes; for example, SDA=X'10'-20 means that recording is to be done on the range of devices with symbolic addresses of decimal 16 (X'10') through decimal 20.

Default: none.

RID
identifies the resource(s) to be traced.

Specified as: a 1 to 17 character name, the first character of which must be alpahbetic. This is the name of the resource or resource range that is to be traced. If two resource names are entered, separated by a hyphen, all the resources having network addresses in between the two named (and including the two named) resources will be traced, provided the two are in the same SUBAREA.

Default: none.

SVC
identifies the SVC number (or range of SVC numbers) to be traced.

Specified as: a decimal integer(s) or as a hexadecimal integer(s).

Default: none.

PI
identifies the program interrupt(s) that are to be traced.

Specified as: a decimal integer(s) or as a hexadecimal integer(s).

Default: none.

EXT
identifies lthe external interrupts that are to be traced.

Specified as: a decimal integer(s) or as a hexadecimal integer(s).

Default: none.

MC
specifies that monitor call recording is to be included in the trace.

Specified as: a decimal integer(s) or as a hexadecimal integer(s). The maximum number of calls that can be traced at any one time is decimal ten.

Default: none.

RTAM
specifies that RTAM tracing is to be done.

Specified as:

*ENT - all RTAM entry points that are called will be included in the trace.

*PIU - all NCP SNA headers (ex: TH + RH...) are included in the trace.

Default: none.

TASK
specifies that tracing is to be done at the task level; if this operand is specified, all operands that precede it in the metalanguage format shown above cannot be specified.

TASKID
identifies the task for which tracing is to be done.

Specified as: a two digit hexadecimal number.

Default: none.

Note: SVC, PI, and EXT at the task level have the same definitions and specifications as those given earlier for the IO level.

RTAM
specifies, at the task level, that all interrupts that the task receives from TAMII are to be recorded.

Specified as: Y or N

Default: N

DSNAME
specifies the name of the data set that will contain the trace data.

Specified as: a 1 to 34 alphanumeric name; the first character must be alphabetic.

Default: see NOREPORT below.

NOREPORT
specifies that all trace data will not be accumulated; instead a circular log will contain the trace data.

Specified as: Y

Default: all trace data will be accumulated in the data set specified by DSNAME above.

REPORT
when issued, whatever is in the circular log will be sent to the terminal; this operand has no meaning if DSNAME is specified.

Default: none.

TRACID
a system supplied identity that permits the programmer to communicate with the system regarding a trace that the programmer had previously initiated.

Specified as: a decimal integer (sent from the system to the programmer after the trace has been started).

Default: none.

END
specifies to the system that a trace is to be ended.

Specified as: Y

Default: none.

Examples 1: to trace all I/O interrupts and I/O instructions issued to a bank of tape drives whose SDAs are X'40' through X'46', enter:

```
gtf sda=x'40-x'46',io=y,dsname=tapegtf
```

This GTF command will cause the following display:

```
GTF TRACE HAS STARTED AND TRACE OD IS - number
```

The number displayed is lthe TRACEID number to be used when the trace is to be ended, as follows:

```
gtf end=y,tracid=number
```

The above command informs the supervisor that the GTF request identified lby 'number' is to be stopped. The supervisor will stop the trace and send back to the issuing task the last filled or partially filled record written to the trace data set. After the record is written, GTF closes the output data set and prompts the user that the specified GTF request has ended.

Examples 2: this example illustrates how to maintain a short log of previous events using the NOREPORT and REPORT operands. Assume a group of 2250s on a single control unit which hangs under certain conditions, and the maintenance person wants a log of the I/O events preceding the hang. A 32000 page data set of trace records is neither needed nor wanted -- the hang may not occur for several hours or even days. To obtain such a log, issue the GTF command with the NOREPORT operand and an

SDA range which included all 2250 SDAs on the particular control unit. Assuming the SDAs were X'30' through X'33', enter:

```
gtf io=y,sda=x'30'-x'33',noreport=y
```

This would cause the supervisor to allocate a 2048 byte block for maintaining a circular log of all I/O events concerning the given SDAs. The size of the entries in the log are 48 bytes in length which means that the log would have a maximum history of 42 entries.

When the 2250 control unit hangs, enter the following command to cause the log to be displayed on the terminal and a fresh log block to be started:

```
gtf report=y,tracid=number
```

Note that number is the trace number displayed as a result of a previously issued GTF command. This causes a display of the circular trace log but does not stop or end the trace. To discontinue the trace, enter:

```
gtf end=y,tracid=number
```

This last command will stop the trace and will also cause the contents of the last trace block to be displayed at the terminal.

MAPGEN -- Create Task Storage Map

The MAPGEN command creates a dataset consisting of a complete storage map for your task. You may obtain VM, RC or both, with or without entry points.

Operation	Operand
MAPGEN	[TYPE=][,LEVEL=][,PRINT=][,EP=][,RUNMODE=]

TYPE
identifies the type of maps desired.

Specified as: RC, VM or ALL

Default: ALL

LEVEL
may be specified as a character string which will be the title of the map dataset developed.

Specified as: a character string

Default: ????????????????????

Execution:

PRINT
specifies whether or not the dataset is printed

Specified as: Y or N.

Default: Y.

EP
specifies whether or not entry points are to be included in the dataset.

Specified as: Y or N.

Default: N.

RUNMODE
specifies whether or not a non-conversational task is to be created to build the map dataset.

Specified as: FORE or BACK.

Default: FORE.

Functional Description: MAPGPV reads the TDY for its task and builds a dataset from the entries. It sorts the names alphabetically and lists VM and RC names separately in the dataset.

MOVEUSER Command

This command moves the data owned by a user and the associated SYSUSE, SYSSVCT, and USERCAT records to a new pool. Also, it will move the user datasets on a pool onto a previously specified subset of PVN numbers within the same pool (the 'compress' option).

Note: all users on a pool must be 'compressed' prior to any partitioning of the pool (see the BLDPOOL command).

Operation	Operands
MOVEUSER	USERID=user to be moved/compressed [,POOLID={name of destination pool|*POOL}] [,CONF=confirmation messages] [,ERASEOLD=erase dataseton old pool after move] [,PAGECHK=check rvns of all data set pages] [,TOUID=destination userid] [,QUITUID=quit user on old pool after move] [,DUPACTON=action if touid catalog already has data set]

Note: MOVEUSER has been written so that the complete status of the move is cataloged before and after any external data operation is taken. MOVEUSER has complete recoverability; it can be restarted from any point, after the initial checking and locking routines have completed initialization. All dataset status information is kept in the catalog.

USERID
identifies the user whose data sets are to be moved; this parameter must be specified.

Specified as: one to eight alphameric characters; the first must be alphabetic.

POOLID
identifies the pool to which the user's datasets are to be moved; this parameter must be specified.

Specified as: one to eight alphameric characters, or *POOL. If *POOL is specified, a SETRVN must have been previously issued for the user, and the user's datasets will be 'compressed' within the

RVN limits of this SETRVN command, within the same pool. (Refer to Notes under the SETRVN command description.)

CONF
indicates whether or not confirmation messages are to be issued as datasets are moved.

Specified as: Y or N

Default: N

ERASEOLD
indicates whether or not data sets are to be erased from their original location after being moved.

Specified as: Y or N

Default: Y

Note: if POOLID=*POOL, ERASEOLD=Y is in force; if TOUID=USERID and POOLID not equal to *POOL, ERASEOLD cannot be specified.

TOUID
identifies the userid in the new pool who will own the datasets being moved.

Specified as: one to eight alphameric characters; the first must be alphabetic.

Default: USERID; i.e., the user will have the same identity in the new pool that he/she had in the old pool.

Note: TOUID can be pre-joined or not, but cannot be joined to any pool except the one specified in POOLID.

Moving to a new pool loses all sharing information, but builds a new, compressed catalog.

If POOLID=*POOL the TOUID parameter is ignored.

PAGECHK
indicates whether or not all dataset pages (rather than the DSCBs) are checked to determine if any lie outside the RVN limits in force for this USERID. This parameter is meaningful only if POOLID=*POOL.

Specified as: Y or N

Default: Y if MULTIVOL=Y; N if MULTIVOL=N.

CAUTION: if MULTIVOL was Y and is now N, PAGECHK=Y must be specified; otherwise, any data pages outside the RVN limits now in force for this USERID will be lost.

QUITUID
indicates whether or not the USERID (in the old pool) is to be QUIT after the datasets have been moved to the new pool.

Specified as: Y or N

Default: Y

Note: if POOLID=*POOL, the QUITUID parameter is ignored. If TOUID=USERID, N cannot be specified. If QUITUID=N (and is valid), all the user datasets will be duplicated under the new userid.

DUPACTON
identifies the action to be performed if a dataset name to be moved already exists in the TOUID's catalog.

Specified as:

SKIP - do not copy the 'from' dataset.

COPY - the TOUID's dataset is erased, then the USERID dataset is copied.

DSCBCHK - a check is made to see if the datasets are similar; if not, then COPY is assumed. When they are similar the change dates are checked: if the 'from' dataset is newer, COPY is assumed; if the TOUID's dataset is newer, SKIP is assumed.

Default: SKIP

Functional description: the program makes checks on the privilege of the issuer, then checks the input parameters for validity. Then the following checks/locks are set:

A. USERID has no currently active tasks, and there is enough space on the destination pool to hold all the USERID datasets. This check is done on a dataset basis for a compress move; i.e., when POOLID=*POOL.

B. the USERID/TOUID is placed in MOVEUSER status.

C. the USERID/TOUID datasets are locked, and are not accessible to any user except the one executing the move.

D. a unique generation data group is cataloged.

E. the USERID/TOUID SYSSVCT records are updated to reflect MOVEUSER status.

F. those datasets to be moved (those outside the RVN limits) are marked in the USERID's catalog.

The initial phase of MOVEUSER is now complete and MOVEUSER enters the 'movedata' phase where the marked datasets are moved. In case the job abends for any reason, it can be restarted by re-issuing the command.

The following is the sequence of events in moving one dataset:

A. a space check is made to see if there is enough space within the destination pool. If there is not enough space, an information message is issued and the next dataset will be moved. After all datasets are moved a check is made to determine if any were skipped: if all were moved, the program goes to the next phase; if any were skipped, there will again be a space check and if there is still not enough space for any dataset, MOVEUSER will exit with an error message leaving the USERID in a MOVEUSER status.

B. The USERID/TOUID datasets will be DDEFed and VV (VAM-TO-VAM) is called to copy the dataset. If there is an abnormal termination during the VV, when MOVEUSER is restarted, the partially copied dataset is erased and the VV is attempted again. If there is a second abnormal termination, the dataset will then be recataloged under the userid performing the move.

C. For TOUID=USERID, the following six steps are performed:

1. The new E DSCB is updated with the original DSNAMES.

2. The original sblock DSCB pointer is changed to point to the new dataset.

3. The old DSCB is updated to the unique DSNAME. The catalog sblock DSCB pointer is changed to the new dataset.

4. The old dataset is erased.

5. The catalog is updated to reflect the move is complete.

6. Confirmation messages are outputted (if CONF=Y).

After all datasets are moved, the 'movedata' phase is completed by updating the catalog to reflect MOVEUSER is now in the JOIN phase.

The JOIN phase performs those items necessary so that the user will be able to logon the new pool after the move is complete. The USERCAT is moved to the new pool, the SYSUSE, SYSSVCT, and CHBUID for the new pool are updated. After the JOIN phase is complete, this information is kept in the SYSSVCT entry of the USERID/TOUID.

When QUITUID=Y, the USERID is quit from the old pool. This consists of deleting the SYSCAT member, erasing USERCAT, deleting SYSSVCT and SYSUSE entries, and deleting the user from the CHBUID of the old pool.

Finally, MOVEUSER goes through a cleanup phase to reset any locks that were set, so that the USERID/TOUID will be able to logon and his/her datasets can be referenced by other users.

NEWMSG (New Updates for Messages) Command

This command brings the most active messages up to date.

Operation	Operand
NEWMSG	

Note: There are no operands.

Functional Description: NEWMSG reinitializes a table (CHBMSG) in which the User Prompter routine maintains the most active messages in the system. All messages in the most-active list are cleared, and any subsequent reference is read into the table from SYSLIB(SYSMLP) and written to the user. The most-active messages in the system are moved into the most-active-message table in the User Prompter's PSECT.

Programming Notes: For standard message processing, the system searches the User Prompter's table for the message before searching SYSMLP. If the message ID is located in the table, the message is issued; the User Prompter does not search through SYSMLP for any message it finds in the buffer. If SYSMLP is edited to change the text of some messages when a copy of the message already exists in the table, the message in the table, existing prior to the update, might be issued.

The NEWMSG command should be issued following any edit of SYSMLP to ensure that current messages are used by the system.

Example: The operator has edited the system message file (SYSMLP). He added several new messages and modified several existing ones. To ensure that all tasks now active in the system will receive current messages, he issues the NEWMSG command.

User: newmsg

System: Clears out the most-active-message table; subsequent references to those messages cause them to be replaced in the buffer table with the current version of the message and written to the user.

PATCLEAR (Clear Page Assignment Table) Command

The PATCLEAR command initializes VAM-formatted disks mounted on private devices. This frees the data pages (including DSCB pages) for reuse.

Operation	Operand
PATCLEAR	DEVICE=device type,VOLID={volume serial number|PRIVATE}, RUNMODE={FORE|BACK} [,PAGING={Y|N}]

DEVICE=
specifies the type of direct access device on which the pack to be initialized is mounted.

Specified as: 2311, 2314, 3330, 333B, or 3350.

VOLID=
identifies the volume to be initialized.

Specified as: one to six alphameric characters indicating the serial number of the volume to be initialized; or PRIVATE if a system scratch pack is to be mounted.

RUNMODE=
indicates that the time-shared PATCLEAR is to be run conversationally or nonconversationally.

Specified as: FORE or BACK.

FORE -- The PATCLEAR is performed in the user's task while running conversationally.
BACK -- When specified in a conversational task, a separate background task is created to perform the PATCLEAR.

Note: RUNMODE is ignored if issued in a nonconversational mode; the PATCLEAR is performed within the background task in which it is issued.

PAGING
indicates whether a pack is to be initialized for data set storage or for paging.

Specified as:

Y - clears the PAT (Page Assignment Table) and indicates that all pages on the pack (except error pages and PAT pages) are to be used for paging.

N - indicates all pages on the pack are to be used for data set storage.

Default: N

CAUTION: The volume to be initialized must be on a private device and may not be in use at the time PATCLEAR is executed. The VOLID operand may not match any VOLID in the public volume table (that is, VOLID can simply be patched, using TSSS). This command can only be executed by tasks having O or P authority, or by the system operator (in the background RUNMODE only).

Functional Description: PATCLEAR zeros the one-byte entries and the relocation entries in the page assignment table (PAT) on VAM-formatted disk packs. The one-byte entries in the PAT for the following pages are not zeroed:

The PAT page(s)
Any error pages
The pages on the error retry cylinder

Programming Notes: The user is responsible for interfacing with system services such as the catalog or external storage allocation; that is, he must use the same precautions he would use in reinitializing the pack with stand-alone DASDI (this is especially important for multivolume data sets).

PATFIX (Fix Page Assignment Table) Command

This command validates entries in the page assignment tables (PATs) by constructing a new PAT from DSCBs that reside on the specified volume, and comparing the reconstructed PAT to the original. A diagnostic report showing any PAT errors or inconsistencies is then printed on the system printer or recorded in a specified data set.

```
┌─────────┬──────────────────────────────────────────────────────────────────┐
│Operation│Operand                                                           │
├─────────┼──────────────────────────────────────────────────────────────────┤
│         │        ┌                          ┐                            │
│         │        │(2311,volume ID,...)      │                            │
│         │        │(2314,volume ID,...)      │                            │
│PATFIX   │ VOLDEF=<(3330,volume ID,...)> [,DEVCOUNT=number]                  │
│         │        │(333B,volume ID,...)      │                            │
│         │        │(3350,volume ID,...)      │                            │
│         │        │PUBLIC                    │                            │
│         │        └                          ┘                            │
│         │                                                                  │
│         │ [,FIX={Y|N}][,REPORTDS=data set name]                            │
│         │ [,DIAGREP=Y|N] [,DAYS=number]                                    │
└─────────┴──────────────────────────────────────────────────────────────────┘
```

VOLDEF=
specifies the volumes for which the PAT entries and DSCBs are to be validated.

Specified as: PUBLIC to validate the PATs in all public storage; 2311, 2314, 3330, 333B, or 3350, followed by a list of private volume serial numbers. Device types cannot be mixed. Public volumes may not be specified by volume serial number.

DEVCOUNT=
specifies the number of direct access devices that are to be used for mounting private volumes; this operand is ignored for public volumes. If the specified number exceeds the number of volumes available, as indicated in the user table, the count from the user table replaces the DEVCOUNT specification. If a specified private volume is found already mounted for the current task, the device on which it is mounted becomes available (if necessary) to PATFIX for mounting other private volumes; in such cases DEVCOUNT is automatically increased by 1.

Specified as: A decimal number.

Default: The number of devices available indicated in the user table.

FIX=
specifies whether or not error conditions in the PAT are to be fixed. If specified as Y, in a conversational task, the system will prompt the user to determine if:

- Erroneous DSCB checksums are to be recomputed.
- Unused DSCB slots having nonzero data are to be zeroed.
- Unused format-F DSCBs are to be zeroed.
- Erroneous format-E DSCBs are to be zeroed.
- The rebuilt PAT page is to replace the original one.

In nonconversational tasks, Y cannot be specified.

Specified as: Y or N

Default: N

REPORTDS=
specifies the name of a previously defined data set into which the PAT diagnostic report is to be written.

Specified as: A fully qualified data set name of a VSAM, VISAM (line data set), or QSAM (without keys) new or old data set.

Default: The report is written, using MSAM, on the system printer.

Note: Only an issuer having privilege class E may take the default.

DIAGREF=
specifies whether or not the 'DATASET UNREFERENCED IN NN DAYS' message is to be written into the report data set. If the user is only interested in looking at the PATFIX output for errors, many 'DATASET UNREFERENCED' messages will make it harder to spot any error messages in the output; this parameter will eliminate them.

Specified as: Y or N.

Default: Y

DAYS=
specifies the number of days the unreferenced messages should be issued for each dataset.

Specified as: a decimal number from 1 to 999.

Default: 30

Functional Description: PATFIX searches the DSCB chains of all the data sets on the specified volumes, and rebuilds the PAT from the DSCB page pointers on those volumes. This search identifies the following error conditions:

- Invalid DSCB entries (for example, DSEENT, DSECHN, DSENME, DSETYP, DSECKS)
- Relocation entry errors
- Unused format-F DSCBs
- Data page entries not accounted for in a DSCB
- Invalid PAT control entries
- Erroneous DSCB entries in PAT

The search also verifies the DSCB page slot count. A comparison between the rebuilt PAT and the original PAT is made to discover any discrepancies between them. If any errors are found and FIX=Y was specified, the user is prompted for the option to recompute (or reset) checksums, zero out unused format-F DSCBs, zero out unused DSCB slots, or to replace the old PAT with the rebuilt PAT. Then, depending on the user's specifications, diagnostic information indicating all errors found in the PATs are written out or the rebuilt PAT replaces the original PAT, or both. The diagnostic information written to the output data set make up five distinct reports.

1. PAT Error Report: Indicates PAT control entries (that is, X'7F', X'C0', or X'FF') are either missing, improperly located, or improperly specified.

2. Data Set Status Report: Lists status information describing every data set on all of the volumes specified by the VOLDEF operand and flags data sets containing diagnostic or error conditions. Status information includes: data set name, date last referred to, change date, total number of pages, volume ID, relative page number of

DSCB slot, and the slot number. Diagnostic and error conditions include: invalid checksums, invalid format-E DSCB, data set not referred to within a specified period, and pages allocated but not used.

3. Shared Data Page Report: Lists the data set name and location of any data sets claiming the same data page (that is, having the same page pointer in their DSCBs).

4. PAT Comparison Data: Lists, for all volumes specified by the VOLDEF operand, any unused format-F DSCBs or data pages.

5. Data Set Profile Report: Lists by user ID and volume ID all data sets and data pages associated with each user ID; they are listed first alphabetically by user ID and then numerically by total number of data pages.

Programming Notes: This command can be used only by system programmers with authority 0.

The normal procedure for using PATFIX is to issue it once, with FIX=N, to produce a diagnostic report, and then to reissue it a with FIX=Y, to fix the error conditions that were indicated in the diagnostic report. If a diagnostic report is requested with the second PATFIX, although some of the reports would have been updated to reflect changes made via FIX=Y (for example, the data set status report would reflect recomputed checksums if the user had requested them), all of the updated PAT changes would not be present in the diagnostic messages (for example, in the PAT comparison report, any data pages that were reclaimed, or unused DSCBs are still listed as error conditions). To get a completely updated diagnostic report, a third PATFIX, with FIX=N, should be issued.

When executing conversationally, the user is prompted, prior to writing out any DSCBs or PATs, to determine if he wants to make any fixes.

Whether public or private volumes are being processed, with FIX=Y, and REPORTDS is also specified, the report data set must reside on a volume type other than one specified by the VOLDEF operand. Thus, if public storage is being processed (with FIX=Y), the report data set must be mounted on a private volume.

When processing public storage, with FIX=Y, this command should not be used while active users are on the system, because all of the PAT entries, for all of the public volumes specified, are locked while PATFIX processing takes place.

If a user asks to write the PAT while in background mode, the writing does take place.

For the report data set, format-U records, and logical record lengths other than 132, will not be processed.

If a data set spans to a volume that was not specified by VOLDEF, the user will be prompted for continuance; if he responds with Y, the volume is added to the volume list specified by VOLDEF, and processing continues.

Examples:

1. For public storage, when FIX is specified as Y and the report data set has a VSAM organization, you might issue:

```
ddef    report,vs,report,volume=(,private),unit=(da,2311)
```

```
patfix  public,,y,report
```

If the report data set is defaulted, the operator must delete a printer and enter:

```
patfix  public,,y
```

2. For a private volume, when FIX is specified as Y and the report data set has a VISAM organization, you might issue:

```
ddef    report,vi,report

patfix  voldef=(2311,dp0575),devcount=1,fix=y,reportds=report
```

POOL? Command

This command displays the status and other information concerning all public storage pools known to the system.

Operation	Operands
POOL?	

Note: there are no operands.

Functional description: this command causes a display of the pool's status (maintenance, marked for deletion, etc.), volume state, and the preferred SYSCAT volume serial number (if any) for each public storage pool.

PRGTF Command

This command translates the captured trace data into unformatted but printable hexadecimal characters.

Operation	Operand
PRGTF	INDSN=dsname1 [,OUTDSN=dsname2] [,*B]

INDSN
: the name of the trace data set to be translated.

  Specified as: a 1 to 35 character name; the first character must be alphabetic.

  Default: none.

OUTDSN
: the name of the output data set that will contain the translated trace data.

  Specified as: a 1 to 35 character name; the first character must be alphabetic.

  Default: if defaulted, the translated data will go to the terminal only.

*B
: specifies that translated data is to go to the terminal as well as dsname2.

Specified as: B

Default: translated data will not go to the terminal.

PRINT Command

In addition to the PRINT command operands available to all users (described in Command System User's Guide), the privilege class E user may choose from several tape printing options (the TAPOPT= operand). These options are described in detail under "Printing ASCII Data Sets" in Part I.

SECURE Command

An explanation and the general format of the SECURE command is contained in Command System User's Guide. The operand choices available to the privilege class E (system monitor) user are described below and are in addition to the operands available to all users. All operands must be entered as keywords; they may not be entered positionally.

PR=
designates the number of printers requested.

Specified as: A one or two digit number.

Default: No printers are reserved.

PC=
designates the number of punches requested.

Specified as: A one or two digit number.

Default: No punches are reserved.

RD=
designates the number of card readers requested.

Specified as: A one or two digit number.

Default: No card readers are reserved.

Programming Note: When reserving devices, all devices must be requested in a single SECURE command statement. For example:

```
secure (ta=2),(pr=1),(rd=2)
```

SETRVN Command

This command allows the pre-joined user and system administrators to set volume limits for the allocation of new datasets, or extensions to existing datasets.

Operation	Operands
SETRVN	[USERID=,LOWRVN=,HIGHRVN=]

USERID
identifies the user for whom RVNs are to be set.

Specified as: a userid, or *ALL, or *CC. *ALL refers to all userids joined to the issuer's pool. *CC allows a system administrator to change/display all userids that match the first two characters of his/her userid.

Default: the current RVNs for the issuer's pool are displayed.

LOWRVN
identifies the lowest relative volume number within the pool on which new datasets or extensions to existing datasets will be allocated for the specified USERID(s).

Specified as: a one to three digit number.

HIGHRVN
identifies the highest relative volume number within the pool on which new datasets or extensions to existing datasets will be allocated for the specified USERID(s).

Specified as: a one to three digit number.

Default: if both LOWRVN and HIGHRVN are defaulted, the current RVNs for the specified USERID(s) are displayed.

Notes: the RVN limits may be expanded but not reduced for any USERID who is already in MOVEUSER status.

Following execution of this command any USERID datasets which may be outside the RVN limits are unaffected; the RVN limits apply only to the extensions of existing datasets or new datasets.

However, if a MOVEUSER command with POOLID=*POOL is executed following a SETRVN, any user dataset with at least a part outside the RVN limits will be copied within the RVN limits (within the same pool) if sufficient space exists; otherwise, a diagnostic is issued. This is the 'compress' option. If a diagnostic is issued indicating insufficient space, several SETRVNs may be issued, each expanding the RVN limits, until sufficient space exists.

TRACE Command

This command will trace the instructions of an NCP/PEP or EP. The trace is simply a record of the instructions in the order in which they were executed.

Operation	Operands
TRACE	resource [,DSNAME=] [,NCPDES=]

resource
identifies the resource to be traced.

Specified as: a 1-8 character name, the first character must be alphabetic. This name must also be the name of a region in the TSS*****.SYSRCS dataset.

Default: There is no default for this parameter.

DSNAME
the name of the dataset that will contain the trace output.

Specified as: a one to thirty-five character name. The first character must be alphabetic. Note: If this parameter is omitted, the output will go only to the terminal.

NCPDES
directs the output to the dataset only or to the dataset and the terminal.

Specified as: LD for output to the dataset only or LDT for output to both the dataset and the terminal.

TRACEND Command

This command will terminate a trace that is active.

Operation	Operands
TRACEND	TRACID=trace identification

TRACID identifies the trace that is to be terminated.

Specified as: a 4 digit number.

Default: There is no default for this parameter.

UPDTUSER (Update User Tables) Command

The UPDTUSER command causes the data about the use of external storage, currently in each user table, to be updated with the information from each user's catalog and DSCB entries.

Operation	Operand
UPDTUSER	[MODE={A|S}]

MODE
specifies whether all or selected user entries in the SYSUSE data set are to be updated.

Specified as: A - all
S - selected (only entries for users with active tasks or owning shared data sets that are being used)

Default: A

Functional Description: This command enables privileged system programmers having an 0 authority code to update all user tables in SYSUSE, with usage statistics from various user catalog entries.

If mode S is specified, only those entries for active users or for those with shared data sets in use at the time of issuing UPDTUSER (or, if the system failed and was restarted, users active at the time of system failure) are updated. A flag (USEADC) is set on in the user entry for active users; if S is specified, only those entries with this flag on are updated.

Programming Notes: UPDTUSER erases temporary public data sets and updates the total number of pages associated with each user table in SYSUSE.

UPDTUSER should be issued after each re-creation of public storage caused by issuing RPS or CVV commands and at any time the user tables are suspected of being incorrect.

If the user table is suspected of errors for active users only, such as might be the case following a system failure and restart, the use of mode S might reduce the updating time.

When the user table of the task issuing the UPDTUSER command is updated, the shared virtual storage record of that user table is updated to correspond to the updated SYSUSE record resulting from execution of UPDTUSER.

EXAMPLE:

User: updtuser

System: Returns the following message to SYSOUT: "nnnn USER TABLE STORAGE ALLOCATIONS UPDATED AGAINST DSCBS".

USAGE Command

See the Command System User's Guide and, for determining usage of the system by users other than the USAGE issuer, the Manager's and Administrator's Guide.

VDMP Command

This command will print on the system SYSOUT device one to all pages of a VAM data set, object text, all the DSCBs for a data set or all the DSCBs on the given volume(s).

Operation	Operand
VDMP	DSNAME=data set name [,CENAME=control section name] [,DSTYPE={DS|OBJ|DSCB}] [,OFFSET=page offset] [,COUNT=number of pages] [,VOLDEF={PUBLIC|(device type,volume identification [,volume identification])}]

DSNAME
identifies the data set to be dumped or the data set of the DSCB to be dumped. The data set must reside on direct-access storage.

VAM data sets must be cataloged unless the VOLDEF parameter is specified.

Specified as: a fully qualified data set name with (optionally) a generation number and/or member name of a VPAM data set. When specified, the member name is enclosed in parenthesis and immediatlely follows the VPAM data set name. When VOLDEF is specified, the DSNAME must be prefixed by the USERID. Otherwise, the USERID is not used. *ALL is used in combination with DSCB and VOLDEF parameters for the given volume(s).

CENAME
identifies the control section or entry point of object text to be dumped. CENAME is to be specified only for DSTYPE=OBJ. When specified, member name is ignored.

Specified as: a control section name.

DSTYPE
specifies the data type to be processed.

Specified as:
DS - data set
OBJ - object text
DSCB - data set control block

Default: OBJ is assumed.

OFFSET
specifies the page offset from the beginning of the data to be dumped. For DSTYPE=DSCB, the OFFSET parameter is ignored.

Specified as: 0 to ($2^{19}-1$) in decimal digits or the equivalent quoted hexadecimal value with preceding 'X'. Arithmetic expressions may be used (maximum of 20 characters). They should be of the same format as TSSS and PCS and should be expressed in pages. See Example 4.

Default: 0 is assumed.

COUNT
identifies the number of pages to be dumped. For DSTYPE=DSCB, the COUNT parameter is ignored.

Specified as: 1 to 20,000 in decimal digits or the equivalent quoted hexadecimal value with preceding 'X'. Arithmetic expressions may be used (maximum of 20 characters) and should be of the same format as TSSS and PCS and should be expressed in pages.

Default: all of the control section, data set or associated DSCBs are assumed.

VOLDEF
identifies specifically the volume(s) to be searched for the data set with a maximum use of two volids. The use of the VOLDEF parameter will cause the catalog entries for the DSNAME to be bypassed.

Specified as:
PUBLIC - for searching all public volumes.
(device type,volume identification [,volume identification]) - for particular volumes or a volume to be searched.
device type - 2311, 2314, 3330, 333B, 3350.
volume identification - a valid volume ID consisting of 1 to 6 characters.

Functional Description: The VDMP command dumps the referenced object text, data set or all DSCBs. The user may specify the dump data set be created on his own private volume (disk or tape) by entering DEFAULTS commands for UNIT, VOLUME and LABEL parameters to be used in the VAMACC DDEF. For a non-conversational task a SECURE command must be issued by the user immediately following the LOGON command. The dump will begin with an appropriate heading. Each line of the dump will contain 32 bytes in hexadecimal character formats. At the beginning of each line will be the displacement. For data sets, displacement indicates the byte offset from the beginning of the page. Displacement indicates the offset from the beginning of the control section for object test and from the beginning of the DSCB for a DSCB dump. When dumping object text, if a null page is encountered, the displacement is incremented by 4096.

CAUTION: The use of the VOLDEF parameter will significantly slow down the search for the data set. When the VOLDEF parameter is specified, the absolute generation name must be used.

Programming Note: Only authority 'O' users may use the VOLDEF parameter. In addition, use of the VOLDEF parameter is to be used only when the data set cannot be accessed in any other way.

For privileged system programmers, *ALL as a DSNAME requests all DSCBs for a given volume. If DSNAME=*ALL, the VOLDEF parameter must be specified.

Authority 'U' will only be able to dump OBJ (object text) while 'O' and 'P' authorities may also dump DS and DSCB.

Example: Example 1 dumps on the system SYSOUT the first 23 pages of the first generation of NEW1 of A.B.C.D. Instead of the catalog, all public volumes will be searched.

```
VDMP    TSS*****.A.B.C.D.G0001V00(NEW1),,DS,,23,PUBLIC
```

Example 2 dumps on the system SYSOUT the first page of the CSECT, TEST of joblib. Instead of the catalog, volumes A and B on a 2314 are searched for the object text.

```
VDMP    VOLDEF=(2314,A,B)COUNT=1,DSNAME=          -
          TSS*****.JOBLIB,CENAME=TEST
```

Example 3 dumps on the system SYSOUT all associated DSCBs of ABC.DEF.

```
VDMP    ABC.DEF,,DSCB
```

Example 4 dumps on the system SYSOUT the fifth page of the data set USER.

```
VDMP    USER,,DS,5,1
```

VDSP Command

This command will display on the user's SYSOUT device up to 2^{29} bytes of data from a VAM data set or up to 10,000 DSCBs from a DSCB chain.

```
|Operation|Operand
|---------|-------------------------------------------------------------
|VDSP     |DSNAME=data set name
|         |[,CENAME={control section name|entry point name}]
|         |[,DSTYPE={DS|OBJ|DSCB}] [,OFFSET=byte offset]
|         |[,COUNT=number of bytes]
|         |[,VOLDEF={PUBLIC|(device type,volume identification
|         |                              [,volume identification])}]
```

DSNAME
identifies the data set to be patched or the data set name of the DSCB to be patched. The data set must reside on direct-access storage. VAM data sets must be cataloged unless the VOLDEF parameter is specified.

Specified as: a fully qualified data set name with (optionally) a generation number and/or member name of a VPAM data set. When specified, the member name is enclosed in parenthesis and immediatlely follows the VPAM data set name. When VOLDEF is specified, the DSNAME must be prefixed by the USERID. Otherwise, the USERID is not used.

CENAME
identifies the control section or entry point of object text to be patched. CENAME is to be specified only for DSTYPE=OBJ. When specified, member name is ignored.

Specified as: a control section name, or an entry point name.

DSTYPE
specifies the data type to be displayed.

Specified as:
DS - data set
OBJ - object text
DSCB - data set control block

Default: OBJ is assumed.

OFFSET
specifies the byte offset from the beginning of the data to be displayed.

Specified as: 0 to (2^{29}-1) in decimal digits or the equivalent hexadecimal value with preceding 'X' if DSTYPE=OBJ or DS. For DSTYPE=DSCB, OFFSET is specified as 0 to (4096 x 633)-1 in decimal digits or the equivalent quoted hexadecimal value with preceding 'X'. Arithmetic expressions may be used for all types (maximum of 20 characters) and they should be of the same format as TSSS and PCS and may be expressed in pages plus the number of bytes into that page. (See Example 4.)

Default: 0 is assumed.

COUNT
identifies the number of bytes to be displayed.

Specified as: 1 to (20,000 x 4096) in decimal digits or the equivalent quoted hexadecimal value with preceding 'X', if DSTYPE=DS or OBJ. For DSTYPE DSCB, COUNT is specified as 1 to (633 x 4096) in decimal digits or the equivalent quoted hexadecimal value with preceding 'X'. Arithmetic expressions may be used for all types (max-

imum of 20 characters) but should be of the same format as TSSS and PCS.

Default: 16 is assumed.

VOLDEF
identifies specifically the volume(s) to be searched for the DSCB with a maximum use of two volids. The use of the VOLDEF parameter will cause the catalog entries for the DSNAME to be bypassed.

Specified as:
PUBLIC - for searching all public volumes.
(device type,volume identification [,volume identification]) - for particular volumes or a volume to be searched.
device type - 2311, 2314, 3330, 333B, 3350.
volume identification - a valid volume ID consisting of 1 to 6 characters.

Functional Description: The VDSP command displays the referenced object text, data set or DSCBs on the user's SYSOUT. Each line displays first the displacement and then 16 bytes in both hexadecimal and character. All unprintable characters will be represented as periods.

For data sets, the displacement field (from the beginning of the page) is displayed as 8 bytes and as 4 bytes for object text. If a null page is encountered within object text, the displacement will be incremented by 4096. For DSCBs, the displacement will be shown in 2 bytes indicating the byte offset from the beginning of the DSCB.

CAUTION: The use of the VOLDEF parameter will significantly slow down the search for the data set. When the VOLDEF parameter is specified, the absolute generation name must be used.

Programming Note: System default for RELEASE will be applied for internal DDEFs. Only authority 'O' users may use the VOLDEF parameter. In addition, use of the VOLDEF parameter is to be used only when the data set cannot be accessed in any other way.

Authority 'U' may display only object text while authorities 'O' and 'P' may also display data sets and DSCBs.

Example: Example 1 displays on the user's SYSOUT 50 bytes of data set ABC starting at the 101st byte.

```
VDSP    ABC,,DS,100,50
```

Example 2 displays on the user's SYSOUT 64 bytes of the type E-DSCB of SOURCE.XYZ starting at the 37th byte. Instead of using the catalog volume DB584 on a 2311 is searched for SOURCE.XYZ.

```
VDSP    TSS*****.SOURCE.XYZ,VOLDEF=(2311,DB584),          -
          DSTYPE=DSCB,OFFSET=X'24',COUNT=X'40'
```

Example 3 displays on the user's SYSOUT the 1st page of object text TEST1 of USERLIB.

```
VDSP    USERLIB,TEST1,COUNT=X'1000'
```

Example 4 displays on the user's SYSOUT the 6th page of object text COMM of USERLIB at a byte offset of 50 into that page.

```
VDSP    USERLIB,COMM,,5 x 4096 + 50VMEREP
```

VPAT -- Command

This command will cause the referenced data set, DSCB or object text, to be updated with up to 50 bytes of data from the user's SYSIN.

```
|Operation|Operand
|---------|-----------------------------------------------------------
|VPAT     |DSNAME=data set name
|         |[,CENAME={control section name|entry point name}]
|         |[,DSTYPE={DS|OBJ|DSCB}] [,OFFSET=byte offset]
|         |[,COUNT=number of bytes],DATA=replacement string
|         |[,VOLDEF={PUBLIC|(device type,volume identification
|         |                              [,volume identification])}]
```

DSNAME
: identifies the data set to be patched or the data set name of the DSCB to be patched. The data set must reside on direct-access storage. VAM data sets must be cataloged unless the VOLDEF parameter is specified.

  Specified as: a fully qualified data set name with (optionally) a generation number and/or member name of a VPAM data set. When specified, the member name is enclosed in parenthesis and immediatley follows the VPAM data set name. When VOLDEF is specified, the DSNAME must be prefixed by the USERID. Otherwise, the USERID is not used.

CENAME
: identifies the control section or entry point of object text to be patched. CENAME is to be specified only for DSTYPE=OBJ. When specified, member name is ignored.

  Specified as: a control section name, or an entry point name.

DSTYPE
: specifies the data type to be processed.

  Specified as:
  DS - data set
  OBJ - object text
  DSCB - data set control block

  Default: OBJ is assumed.

OFFSET
: specifies the byte offset from the beginning of the data to be patched.

  Specified as: 0 to ($2^{29}-1$) in decimal digits or the equivalent hexadecimal value with preceding 'X' if DSTYPE=OBJ or DS. For DSTYPE=DSCB, OFFSET is specified as 0 to (4096 x 633)-1 in decimal digits or the equivalent quoted hexadecimal value with preceding 'X'. Arithmetic expressions may be used for all types (maximum of 20 characters) and they should be of the same format as TSSS and PCS and may be expressed in pages plus the number of bytes into that page. (See Example 4.)

  Default: 0 is assumed.

COUNT
: identifies the number of bytes to be patched.

Specified as: 1 to 50 in decimal or the equivalent quoted hexadecimal value with preceding 'X'. Arithmetic expressions may be used (maximum of 20 characters) and be of the same format as TSSS and PCS.

Default: the length of the data field is assumed.

DATA
designates the data string that is to be the replacement.

Specified as: a quoted hexadecimal or character string with a preceding 'X' or 'C'.

VOLDEF
identifies specifically the volume(s) to be searched for the DSCB with a maximum use of two volids. The use of the VOLDEF parameter will cause the catalog entries for the DSNAME to be bypassed.

Specified as:
PUBLIC - for searching all public volumes.
(device type,volume identification [,volume identification]) - for particular volumes or a volume to be searched.
device type - 2311, 2314, 3330, 333B, 3350.
volume identification - a valid volume ID consisting of 1 to 6 characters.

Functional Description: The VPAT command replaces on the disk up to 50 bytes with the data entered on the user's SYSIN device. If the length of the entered data string is different from the length specified in the COUNT parameter, it will be padded or trunctaed. Character data will be padded with blanks or truncated on the right; hexadecimal data will be padded with zeros or truncated on the left. If DSTYPE=DSCB, the CHECKSUM will be automatically recomputed.

CAUTION: The use of the VOLDEF parameter for DSTYPE=DSCB will significantly slow down the search for the DSCB.

Since no check will be performed for the validity of the patch, the user should exercise extreme caution. Patching across DSCB boundaries is not allowed.

Programming Note: Only authority 'O' may patch DSCBs, DS and OBJ while 'U' authorities may patch OBJ and 'P' authority may patch DS and OBJ. Only authority 'O' users may patch system data sets.

Only authority 'O' users may use the VOLDEF parameter with DSTYPE=DSCB. In addition, use of the VOLDEF parameter is recommended only when the DSCB cannot be accessed in any other way. The patching of shared VPAM POD is not allowed.

The system default for REVIEW is applied for the display of the string prior to the patch. If REVIEW=Y, the user is prompted for continuation. System default for RELEASE will be applied for internal DDEFs.

Example: Example 1 replaces the first 3 bytes of the 533rd page of the data set XYZ with the character string 'DEF':

```
VPAT   XYZ,,DS,X'214000',3,C'DEF'
```

Example 2 replaces the first 4 bytes of the 3rd DSCB associated to ATTN1 with X'00040301':

```
VPAT   DSNAME=ATTN1,,DSCB,X'200,X'040301'
```

Example 3 patches the object text at entry point EP1 of the member TEST in USERLIB. It replaces 3 bytes starting 4098 bytes past entry point EP1 with ABC:

```
VPAT   USERLIB,EP1,,4098,3,X'C1C2C3'
```

Example 4 patches the 5th page of a data set NEW at 100 bytes into that page:

```
VPAT   NEW,,DS,4 x 4096+100,3,C'MNO'
```

APPENDIX A: SYSTEM ENTER CODE TABLE

Decimal	Hex	Name	ENTRY POINT	PSECT
		TAMII/MTT/PPLI		
0	00	READ/WRITE	CZCYM1	CZCYMP
1	01	BATCH MONITOR	CZABAE	CZABAE
2	02	GATE MACROS	CZFTAU	CZFTPP
3	03	READQ	CZCTC3A	CZFTPP
4	04	WRITEQ	CZCTC4A	CZFTPP
5	05	FINDQ	CZCTC2A	CZFTPP
6	06	FREEQ	CZCTC6A	CZFTPP
7	07	ATTENTION	CZFAA1	CZFAAP
8	08	TERMPRO	CZFTE15	CZFTPP
9	09	PPLI ROUTINES	CZPPL1	CZPPLP
10	0A	MTT/MTTDCN	CZFAH3	CZFAHP
12	0C	OPNDST/CLSDST	CZFTF2	CZFTFP
14	0E	GETDV	CZATM2	CZATMP
15	0F	SETDV	CZATM1	CZATMP
		INTERRUPT HANDLING		
16	10	SIR	CZCJSA	CZCJSP
17	11	DIR	CZCJDA	CZCJDP
18	12	INTINQ	CZCJIA	CZCJIP
19	13	STIMER/TTIMER	CZCJA1	CZCJAR
		SAM		
32	20	READ/WRITE	CZCRAS	CZCRAP
33	21	CHECK	CZCRCS	CZCRCP
34	22	CNTRL	CZCRBS	CZCRBP
36	24	POINT	CZCRMA	CZCRMP
37	25	BSP	CZCRGA	CZCRGP
		VM ALLOCATION		
48	30	GETMAIN (R)	CZCH2	CZCG5
49	31	GETMAIN (PAGE)	CZCG2	CZCG5
50	32	FREEMAIN (R)	CZCH3	CZCG5
51	33	FREEMAIN (PAGE)	CZCG3	CZCG5
		VAM		
56	38	VDMEP	CZCQK1	CZCQKP
57	39	DUPOPEN	CZCEY1	CZCEYP
58	3A	DUPCLOSE	CZCEZ1	CZCEZP
61	3D	VISAM SETL	CZCPC3	CZCPC3
62	3E	VSAM PUT	CZCOS3	CZCOS3
63	3F	LIBESRCH	CZCDL3	CZCDLP
64	40	READ/WRITE	CZCPE1	CZCPEP
65	41	ESETL	CZCPD1	CZCPIP
66	42	RELEX	CZCPG1	CZCPIP
67	43	DELREC	CZCPH1	CZCPHP
68	44	FIND	CZCOJ1	CZCOJP
69	45	STOW	CZCOK1	CZCOKP
70	46	ADD DIRECTORY ENTRY	CZCPL1	CZCPLP
71	47	GETPAGE	CZCPI1	CZCPIP
72	48	INSERT PAGE	CZCOD1	CZCODP
73	49	DELETE PAGE	CZCOD2	CZCODP
74	4A	VSAM PUT EXTERNAL USER	CZCOS1	CZCOS1
75	4B	VSAM PUT INTERNAL	CZCOS2	CZCOS2
76	4C	MOVEPAGE	CZCOC1	CZCOCP
77	4D	FLUSHBUF	CZCOV1	CZCOVP
78	4E	VISAM GET PAGE INPUT	CZCPI2	CZCPIP
79	4F	VISAM GET PAGE OUTPUT	CZCPI3	CZCPIP

Figure 38. System ENTER code table (part 1 of 2)

Decimal	Hex	Name	ENTRY POINT	PSECT
		MACRO COMMAND LANGUAGE		
80	50	GATRD/GATWR	CZATC2	CZATCP
81	51	WTO	CZABQ1	CZABQR
82	52	WTOR	CZABQ1	CZABQR
83	53	ERASE	CZAEJ7	CZAEJR
84	54	DDEF	CZAEA3	CZAEAR
85	55	CDD	CZAFS2	CZAFSR
86	56	ABEND	CZACP1	CZACPR
87	57	CPU	CZABD7	CZABDR
88	58	WT	CZABD9	CZABDR
89	59	PR	CZABD3	CZABDR
90	5A	CAT	CZAEI2	CZAEIR
91	5B	DEL	CZAEJ5	CZAEJR
92	5C	COPYDS	CZAFV2	CZAFVR
94	5E	WTL	CZABQ1	CZABQR
95	5F	USATT	CZASA6	CZASAP
96	60	FINDJFCB	CZAEB1	CZAEBR
97	61	CLATT	CZASA7	CZASAP
98	62	REL	CZAFJ2	CZAFJR
99	63	USAGE	CZAGB1	CZAGBP
100	64	FINDDS	CZAEC1	CZAECR
101	65	MSGWR	CZAAD3	CZAADR
102	66	UPDTUSER	CZAGC2	CZAGCR
110	6E	GRAPHIC BUFFERS	CZCVB1	CZCVBP
111	6F	GRAPHIC I/O	CZCVE1	CZCVEP
		GENERAL SERVICES		
112	70	IOREQ	CZCSB1	CZCSBR
113	71	MSAM READ/WRITE	CZCMF1	CZCMFP
114	72	MSAM - SET UNIT RECORD	CZCMD1	CZCMDP
115	73	MSAM FINISH	CZCMH1	CZCMHP
116	74	TSAM READ/WRITE/FINISH	CZCYC3	CZCYC3
128	80	OLTAM - DEV. ALLOC.	CZATG1	CZATGP
129	81	OLTAM - EX. I/O	CZATA1	CZATAP
130	82	OLTAM - POSTING	CZATB1	CZATBP
131	83	OLTAM - TEST COMMAND	CZATS1	CZATSP
144	90	OPEN	CZCLA0	CZCLAB
145	91	CLOSE	CZCLBC	CZCLBP
146	92	FEOV	CZCLDF	CZCLDB
147	93	RFR	CZASD3	CZASDP
148	94	GDV	CZASDX	CZASDP
149	95	AETD	CZASB5	CZASBP
150	96	OBEY	CZASA4	CZASAP
151	97	MCAST	CZATU1	CZATUP
152	98	SYSIN	CZASC7	CZASCP
153	99	LPCINIT	CZASW1	CZAMZP
154	9A	LPCEDIT	CZASW4	CZAMZP
155	9B	PRMPT	CZATS1	CZATJP
156	9C	ATTN	CZASB2	CZASBP
157	9D	GATE	CZATC2	CZATCP
158	9E	ENTRFR	CZASD5	CZASDP
159	9F	DELENT	CZASD6	CZASDP
160	A0	CSTORE	CZCKZ1	CZCKZP
161	A1	NXTRFR	CZASD4	CZASDP
162	A2	DICTIONARY HANDLER	CZASD2	CZASDP
		FORTRAN		
164	A4	FTN TRACEBACK	CZCDT1	CZCDTP
191		Reserved for TSS users.		
254				

Figure 38. System ENTER code table (part 2 of 2)

APPENDIX B: VIRTUAL AND REAL MEMORY SVCS

In this appendix, the DCLASS setting shows what DCLASS operand is required when assembling the macro instruction. Where no DCLASS setting is shown, there is no requirement; that is, the macro definition does not test for a DCLASS setting.

NON-PRIVILEGED PROGRAM SERVICE SVCS

SVC CODE DEC	SVC CODE HEX	MACRO	FUNCTION	DCLASS	CODE RQMT
0-99	00-63	...	RESERVED FOR PROBLEM PROGRAMS	...	...

PRIVILEGED PROGRAM SERVICE SVCS

SVC CODE DEC	SVC CODE HEX	MACRO	FUNCTION	DCLASS	CODE RQMT
100-115	64-73	...	NOT DEFINED	...	...
116	74	EXIT	NORMAL PROGRAM END	USER	NP
117	75	RAESVC	RESTORE AND ENABLE INTERRUPTS	USER/ PRIV	NP,P
118	76	CLIP	READ COMMAND FROM SYSIN (UNCONDITIONAL)	USER	NP
119	77	CLIC	READ COMMAND FROM SYSIN (CONDITIONAL)	USER	NP
120	78	RSPRV	RESTORE PRIVILEGE	USER	NP
121	79	ENTER	ENTER PRIVILEGED ROUTINE	...	NP
122	7A	RTRN	ENTER COMMAND LANGUAGE TO END RUN	...	NP
123	7B	DELET	ENTER DELETE PROGRAM	...	NP,P
124	7C	...	NOT DEFINED	...	...
125	7D	PCSVC	ENTER PCS	PRIV	NP
126	7E	...	NOT DEFINED	...	...
127	7F	DLINK	ENTER DYNAMIC LOADER TO RESOLVE EXTERNAL SYMBOL	...	NP,P

Figure 39. Virtual and Real Memory SVCs (part 1 of 4)

REAL MEMORY PROGRAM SERVICE SVCS

SVC CODE DEC	HEX	MACRO	FUNCTION	DCLASS	CODE RQMT
128-143	80-8F	...	RESERVED FOR INSTALLATION USE	...	...
144-158	90-9E	...	RESERVED FOR TSSS	...	...
159	9F	...	VSS 'AT' IN NON-SHARED VM	...	NP,P
160	A0	...	LOGON MSP	...	P
161	A1	...	DISCONNECT MSP	...	P
162	A2	...	ACTIVATE VSS	...	P
163	A3	...	VSS 'AT' COMPLETE	...	NP,P
164	A4	...	VSS 'AT' IN SHARED VM	...	NP,P
165	A5	...	GET REAL PAGE	...	P
166	A6	...	SHARED PAGE DETERMINATION	...	P
167-169	A7-A9	...	RESERVED FOR TSSS	...	...
170-179	AA-B3	...	NOT DEFINED	...	...
180	B4	RSEG	RESERVE SEGMENT	...	NP,P
181	B5	RELSEG	RELEASE SEGMENT	...	NP,P
182	B6	DSEG	DISCONNECT NAMED SEGMENT	...	NP,P
183	B7	CSEG	CONNECT NAMED SEGMENT	...	NP,P
184	B8	DELSEG	DELETE NAMED SEGMENT	...	NP,P
185	B9	ESEG	EXCHANGE SEGMENT	...	NP,P
186	BA	GPSEG	GET/PUT NAMED SEGMENT	...	NP,P
187	BB	QFLOW	EXTRACT FLOW INFORMATION	...	P
188	BC	SETCTL	SET CONTROL REGISTERS	PRIV	NP
189	BD	XTRCTL	EXTRACT CONTROL REGISTERS	...	NP
190	BE	RTTCTL	REAL TIME CONTROL	PRIV	P
191	BF	...	NOT DEFINED	...	...
192	C0	...	GTF TRACE REQUEST	...	P
193	C1	SAMPLE	SAMPLE SST	...	P
194	C2	ZEROSST	ZERO SST	...	P
195	C3	ATTACH	ATTACH TASK TO SYSTEM	...	NP,P
196-199	C4-C7	...	RESERVED FOR PERFORMANCE MEASUREMENT	...	...
200	C8	...	UPDATE SYS OPERATOR TABLE	...	P

Figure 39. Virtual and Real Memory SVCs (part 2 of 4)

REAL MEMORY PROGRAM SERVICE SVCS

SVC CODE DEC	HEX	MACRO	FUNCTION	DCLASS	CODE RQMT
201	C9	...	NOT DEFINED	...	...
202	CA	TAMSVC	MULTI FUNCTION TAMII SVC	...	P
203	CB	CKALOC	CHECK MTT TERMINAL STATUS	PRIV	P
204	CC	WAIT	WAIT FOR EXTERNAL STIMULI	...	P
205	CD	LCONN	TAMII TERMINAL CONNECT	...	P
206	CE	SCRTSI	SPECIAL CREATE TSI	PRIV	P
207	CF	CONN	CONNECT AN MTT TASK	...	P
208	D0	DCON	DISCONNECT AN MTT TASK	...	P
209	D1	XTRTM	EXTRACT TASK TIME	...	NP,P
210	D2	SETAE	SET ASYNCHRONOUS ENTRY	...	P
211	D3	SPATH	SET I/O DEVICE PATH	PRIV	P
212	D4	ADSBA	TAMII NCP SUBAREA CONTRL	...	P
213	D5	XTRXTS	EXTRACT FROM XTSI	...	NP,P
214	D6	SETXTS	SETUP XTSI	PRIV	P
215	D7	XTRSYS	EXTRACT FROM SYSTEM TABLE	...	NP,P
216	D8	SETSYS	SETUP SYSTEM TABLE	PRIV	P
217	D9	SETTR	SET REAL-TIME INTERVAL	PRIV	P
218	DA	REDTIM	READ TIME OF DAY	...	NP,P
219	DB	ATCS	TAMII I/O REQUEST	...	P
220	DC	...	RMS MODE SET	...	P
221	DD	RESET	RESET SUPRESS DEVICE FLAG	...	P
222	DE	PURGE	PURGE I/O OPERATIONS	...	P
223	DF	...	SET/RESET IMMEDIATE RECORDING FLAG	...	P
224	E0	ROPAGE	READ ONLY PAGE UPDATE	...	P
225	E1	...	NOT DEFINED	...	...
226	E2	PULSE	PULSE SCHEDULE LEVEL	...	NP,P
227	E3	CHANGE	CHANGE SCHEDULE LEVEL	...	NP,P
228	E4	SYSER	VM SYSTEM ERROR	PRIV	P

Figure 39. Virtual and Real Memory SVCs (part 3 of 4)

REAL MEMORY PROGRAM SERVICE SVCS

SVC CODE DEC	HEX	MACRO	FUNCTION	DCLASS	CODE RQMT
229	E5	TWAIT	WAIT FOR TERMINAL I/O	...	NP,P
230	E6	AUXPG	EXTRACT AUX PAGE COUNTS	...	NP,P
231	E7	IOCAL	I/O CALL	PRIV	P
232	E8	...	RJE LINE CONTROL	...	P
233	E9	RMDEV	REMOVE DEVICE FROM TASK	...	P
234	EA	ADDEV	ADD DEVICE TO TASK	...	P
235	EB	SETUP	SETUP TSI	PRIV	P
236	EC	ADSPG	ADD SHARED PAGES	PRIV	P
237	ED	DSSEG	DISCONNECT SHARED SEGMENT	PRIV	P
238	EE	CNSEG	CONNECT SHARED SEGMENT	PRIV	P
239	EF	EXPND	EXPAND PAGE	...	P
240	F0	VSEND	INTER-TASK COMMUNICATION	...	NP,P
241	F1	CKCLS	CHECK PROTECTION CLASS	...	NP,P
242	F2	PGOUT	PAGE OUT	PRIV	P
243	F3	TSEND	FORCE TIME SLICE END	PRIV	P
244	F4	SETXP	SET EXTERNAL PAGE TABLE	PRIV	P
245	F5	MOVXP	MOVE PAGE TABLE ENTRIES	PRIV	P
246	F6	XTRCT	EXTRACT TSI	...	NP,P
247	F7	QSVC	ENQ/DEQ	...	...
248	F8	AWAIT	WAIT FOR INTERRUPT	...	NP,P
249	F9	DELPG	DELETE PAGE	PRIV	P
250	FA	ADDPG	ADD PAGE	...	NP,P
251	FB	SETTU	SET USER TIMER	PRIV	P
252	FC	DLTSI	DELETE TSI	PRIV	P
253	FD	CRTSI	CREATE TSI	PRIV	P
254	FE	ERROR	RM SYSTEM ERROR	...	...
254	FE	LVPSW	LOAD VIRTUAL PSW	PRIV	P
255	FF	...	NOT DEFINED	...	...

Figure 39. Virtual and Real Memory SVCs (part 4 of 4)

The resident supervisor must pass back to the virtual storage error processors a code identifying the type of software error detected by the supervisor. To accomplish this, the Supervisor Processor must put a GQE on the appropriate task's TSI program interruption queue of the task in error. The interruption code in the GQE contains a value that identifies the cause of the program interruption.

Hexadecimal codes 01 through 13, 40, and 80 are used by the hardware. Codes 14 through 1F are reserved for hardware. The remaining codes through FFFF (including 00) are used for specifying software program interruption errors.

The defined codes are shown in Figure 40.

PI CODE	SVTY CODE	MODULE	ERROR DESCRIPTION
00	3	-	NOT DEFINED
01-1F	-	-	SPECIFIED IN 'PRINCIPLES OF OPERATION'
20-21	3	-	NOT DEFINED
22	3	CEAA0	PAGE LIST LENGTH TOO LONG
		CEAA1	PAGE LIST LENGTH TOO LONG
23	3	CEAA0	NON-EXISTENT BUFFER PAGE
		CEAA1	NON-EXISTENT BUFFER PAGE
24	3	CEAA0	TASK HAS NO DEVICES ASSIGNED
		CEAA1	TASK HAS NO DEVICES ASSIGNED
25	3	CEAA0	IORCB LENGTH EQUALS ZERO
26	3	-	NOT DEFINED
27	1	CEAAF	COUNTER OVERFLOW FOR PROGRAM INTERRUPTS
28	1	CEAAF	COUNTER OVERFLOW FOR SVC INTERRUPTS
29	1	CEAAF	COUNTER OVERFLOW FOR EXTERNAL INTERRUPTS
2A	1	CEAAF	COUNTER OVERFLOW FOR ATTENTION INTERRUPTS
2B	1	CEAAF	COUNTER OVERFLOW FOR TIMER INTERRUPTS
2C	1	CEAAF	COUNTER OVERFLOW FOR I/O INTERRUPTS
2D	1	CEAAF	UNCLASSIFIED TASK INTERRUPT
2E	3	CEAA0	IORCB LENGTH GREATER THAN 4096 BYTES
2F	3	CEAA1	IORCB CROSSES PAGE BOUNDARY
30	3	CEAA0	DEVICE NOT ASSIGNED TO TASK
		CEAA1	DEVICE NOT ASSIGNED TO TASK
31	3	CEAN*	DELETE PAGE OF WRONG CLASS
32	3	CEAA0	NON-EXISTENT SVC PAGE
		CEAA1	NON-EXISTENT SVC PAGE

Figure 40. TSS Extended Program Interrupt Codes (1 of 5)

PI CODE	SVTY CODE	MODULE	ERROR DESCRIPTION
33	3	CEAA1	SVC PAGE NOT IN MAIN STORAGE
34	3	CEAA0 CEAA1	CCW LIST OUTSIDE OF SVC PAGE PGOUT REQUEST MIXES SHARED AND PRIVATE
35	3	CEAND	DELETE PAGE IN UN-ASSIGNED SEGMENT
36	3	CEAND	DELETE UN-ASSIGNED PAGE
37	3	CEAND	INVALID INPUT PARAMETERS TO DELETE PAGE
38	3	CEAND	INVALID RANGE FOR SHARED DELETE
39	3	CEAH7	ATTEMPT TO RE-ASSIGN AN IVM PAGE
3A	3	CEAH7	PAGE NOT IN CALLER'S PAGE TABLE
3B-3C	3	-	NOT DEFINED
3D	3	CEAQ6	THE SHARED SEGMENT TABLE OVERFLOWED
3E-3F	3	-	NOT DEFINED
40	-	-	MONITOR CALL HARDWARE INTERRUPT
41-47	3	-	NOT DEFINED
48	3	CEAH2	INVALID INPUT PARAMETER TO SETUP/XTRCT
49	3	CEAP7	AWAIT SVC NOT EXECUTED REMOTELY OR ELSE NOT ON THE LAST HALFWORD OF AN ECB
4A	3	CEAQ7	INVALID INPUT PARAMETERS TO CONNECT
4B	1	CEAQ5	VSEND SVC NOT EXECUTED REMOTELY
4C	3	CEAQ5	VSEND MCB EXCEEDS 1912 BYTES OR CROSSES PAGE BOUNDARY
4D-4F	3	-	NOT DEFINED
50	3	CEAHQ CEAR3	TASK NOT OF SUFFICIENT PRIVILEGE TO ISSUE SVC TASK NOT OF SUFFICIENT PRIVILEGE TO ISSUE SVC
51	3	CEAH7	SETXP SVC NOT ON FULLWORD BOUNDARY
52	3	CEAH7 CEHDB CEHDE	COUNT OF EXTERNAL ADDRESSES IS ZERO INVALID VMA PASSED TO VSS GET REAL PAGE INVALID TYPE REQUESTED FOR VSS EXIT
53	3	CEAH7	PARAMETER LIST CROSSES PAGE BOUNDARY OR PAGE NOT IN CALLER'S PAGE TABLE
54	3	CEAH7	COUNT OF EXTERNAL ADDRESSES EXCEEDS 1022
55	3	CEAH7	A SPECIFIED PAGE IS UN-ASSIGNED
56	3	CEAH7	EXTERNAL DEVICE ERROR

Figure 40. TSS Extended Program Interrupt Codes (2 of 5)

PI CODE	SVTY CODE	MODULE	ERROR DESCRIPTION
57	3	-	NOT DEFINED
58	3	CEAQ8	INVALID INPUT PARAMETER TO DISCONNECT
59	3	CEANE	INVALID INPUT PARAMETER TO ADD PAGE
5A	3	CEAQ7	ATTEMPT TO CONNECT TO UN-ASSIGNED PAGE
5B	3	CEAKR CEAP0	ATTEMPT TO CANCEL NON-EXISTENT TIMER ATTEMPT TO MOVE FROM UN-ASSIGNED PAGE
5C	3	CEAP0	ATTEMPT TO MOVE TO UN-ASSIGNED PAGE
5D	3	CEAS2	INVALID INPUT PARAMETER TO SETSYS/XTRSYS
5E	3	CEAS4	INVALID INPUT PARAMETER TO SETXTS/XTRXTS
5F	3	CEAP0	MOVE FROM OR TO SHARED PAGE
60	3	CEANE CZCJT	ADD PAGE REQUEST NOT SATISIFIED ENTER SVC ISSUED WHILE IN TYPE III LINKAGE
61	3	CZCJT	ENTER SVC ISSUED WITH INVALID ENTER CODE
62	3	CZCJT	SVC ISSUED IN NON-PRIVILEGED STATE AND NO INTERRUPTION ROUTINE SPECIFIED
63	3	CZCJT	NO ERROR ROUTINE DEFINED FOR DEVICE WITH ERROR
64	3	CZCJT	ASYNCHRONOUS INTERRUPT RECEIVED BUT NO DE AVAILABLE FOR DEVICE
65	3	CZCJT	SETTR NOT ACCEPTED BECAUSE SYSTEM LIMIT
66	3	CZCJT	SVC INTERRUPT RECEIVED WHILE IN TYPE III LINKAGE
67	3	CZCJT	PROGRAM INTERRUPT RECEIVED WHILE IN TYPE III LINKAGE
68	3	CEAQ2	ATTEMPT TO SET TIMER BEYOND 55,364,812 MILLI-SECONDS
69	3	CEAAC	INVALID SDA DETECTED IN ADD DEVICE
6A	3	CEAAK CEAP0	INPUT SDA OUT OF RANGE INVALID INPUT PARAMETERS TO MOVE PAGE
6B	3	CEAQ4	INVALID INPUT PARAMETERS TO CHECK CLASS
6C	3	CEAA1	PAGE OUT REQUEST FOR ZERO PAGES
6D	3	CEAQ6	INVALID INPUT PARAMETERS TO ADD SHARED PAGE
6E-6F	3	-	NOT DEFINED
70	3	CEAAK	A SETAE WAS ISSUED TO DEVICE NOT ASSIGNED TO TASK
71	3	CEAAK	A SETAE WAS ISSUED SPECIFYING A NON-EXISTENT TASK

Figure 40. TSS Extended Program Interrupt Codes (3 of 5)

PI CODE	SVTY CODE	MODULE	ERROR DESCRIPTION
72	3	CEAP1	INVALID INPUT PARAMETERS TO EXPAND PAGE
73	3	CEAP1	TASK EXCEEDED MAXIMUM PAGE TABLE PAGES
74-78	3	-	NOT DEFINED
79	3	CEAHQ	INVALID SVC CODE
7A-7B	3	-	NOT DEFINED
7C	3	CEAA0	IOCAL SVC CCW LIST CANNOT BE RELOCATED
7D	1	CEAA0	DRAM CCW LIST CANNOT BE RELOCATED
7E-7F	3	-	NOT DEFINED
80	-	-	PROGRAM EVENT RECORDING HARDWARE INTERRUPT
81-8F	3	-	NOT DEFINED
90	2	CEAAQ	RELOCATION READ: NO PATH AVAILABLE
91	2	CEAAQ	RELOCATION READ: I/O ERROR ON PERMANENT VOLUME
92	2	CEAAQ	RELOCATION READ: I/O ERROR ON MOVEABLE VOLUME
93	3	CEAAQ	RELOCATION READ: SURFACE ERROR
94	2	CEAAQ	RELOCATION READ: START I/O FAILURE
95	2	CEAAQ	SUPERVISOR PAGING REQUEST: NO PATH AVAILABLE
96	2	CEAAQ	SUPERVISOR PAGING REQUEST: I/O ERROR ON PERMANENT VOLUME
97	2	CEAAQ	SUPERVISOR PAGING REQUEST: I/O ERROR ON MOVEABLE VOLUME
98	3	CEAAQ	SUPERVISOR PAGING REQUEST: SURFACE ERROR
99	3	CEAAQ	SUPERVISOR PAGING REQUEST: START I/O FAILURE
9A-9E	3	-	NOT DEFINED
9F	2	CEAAQ	TWAIT READ: NO PATH AVAILABLE
A0	2	CEAAQ	TWAIT READ: I/O ERROR ON PERMANENT VOLUME
A1	2	CEAAQ	TWAIT READ: I/O ERROR ON MOVEABLE VOLUME
A2	2	CEAAQ	TWAIT READ: SURFACE ERROR
A3	2	CEAAQ	TWAIT READ: START I/O FAILURE
A4-AF	3	-	NOT DEFINED
B0	3	CEAP2 CEAP4 CEAP5 CEAP8 CEAQ0	SVC NOT EXECUTED REMOTELY SVC NOT EXECUTED REMOTELY SVC NOT EXECUTED REMOTELY SVC NOT EXECUTED REMOTELY SVC NOT EXECUTED REMOTELY

Figure 40. TSS Extended Program Interrupt Codes (4 of 5)

PI CODE	SVTY CODE	MODULE	ERROR DESCRIPTION
B1	3	CEAP2	SVC NOT ON FULLWORD BOUNDARY
		CEAP4	SVC NOT ON FULLWORD BOUNDARY
		CEAP5	SVC NOT ON FULLWORD BOUNDARY
		CEAP8	SVC NOT ON FULLWORD BOUNDARY
		CEAQ0	SVC NOT ON FULLWORD BOUNDARY
B2	3	CEAP2	PARAMETER LIST CROSSES PAGE BOUNDARY
		CEAP4	PARAMETER LIST CROSSES PAGE BOUNDARY
		CEAP5	PARAMETER LIST CROSSES PAGE BOUNDARY
		CEAP8	PARAMETER LIST CROSSES PAGE BOUNDARY
		CEAQ0	PARAMETER LIST CROSSES PAGE BOUNDARY
B3	3	CEAQ0	INVALID GET/PUT NAMED SEGMENT INDICATOR
B4-BF	3	-	NOT DEFINED
C0	3	CZCJT	ISA DESTROYED
C1-C6	3	-	NOT DEFINED
C7	3	CMABA	UNCORRECTED MACHINE CHECK DURING TASK
C8	3	CEAHQ	TASK HAS EXCEEDED IT'S TSEND SVC MAXIMUM
C9	3	-	NOT DEFINED
CA	3	CEHER	MAJOR VIRTUAL MEMORY SYSER
CB-CF	3	-	NOT DEFINED
D0	3	CEATB	SVC NOT REMOTELY EXECUTED
D1	3	CEATB	INVALID RLN OR NO TERMINAL CONNECTED TO TASK
D2	3	CEATB	INVALID REQUEST CODE
D3	3	CEATB	VALID RLN BUT NO TCT AND REQUEST NOT FREE
D4	3	CEATB	INVALID FLAGS IN TCLEAR REQUEST
D5	3	CEATB	INVALID READ LENGTH
D6	3	CEATB	INVALID WRITE LENGTH
D7	3	CEATB	INVALID DATA ADDRESS FOR WRITE
D8	3	CEATD	SVC NOT REMOTELY EXECUTED
D9	3	CEATD	INVALID RLN IN TAMSVC REQUEST
DA	3	CEATD	INVALID REQUEST CODE IN TAMSVC REQUEST
DB	3	CEATD	ZERO PAGE COUNT IN SAVBFP REQUEST
DC	3	CEATD	INVALID VMA IN SAVBFP REQUEST
DD	3	CEATD	ZERO PAGE COUNT IN RSTBFP REQUEST
DE	3	CEATD	INVALID VMA IN RSTBFP REQUEST
DF	3	CEATD	RSTBFP BUFFER PAGES INCORRECTLY FORMATTED
E0	3	CEATD	RSTBFP BUFFER CONTAINS INVALID DATA
E1	3	CEATD	INVALID VMA IN SETTCT REQUEST
E2	3	CEDMOX	INVALID I/O REQUEST ISSUED BY TAMII
E3	3	CEATB	MORE THAN 248 REQUESTS QUEUED ON TERMINAL
E4-EF	3	-	RESERVED FOR TAMII
F0-FF	3	-	NOT DEFINED

Figure 40. TSS Extended Program Interrupt Codes (5 of 5)

Every task has a task dictionary (TDY), which contains, in addition to the program module dictionaries (PMDs), the hash tables used by the dynamic loader to process external definitions (DEFs) and external references (REFs). The hash table is split into three parts: privileged system, nonprivileged system, and user symbols (see Figure 41). When the loader encounters a REF in a control section with the attribute of PRVLGD, it searches the privileged system hash table; if the attribute of the control section is nonprivileged SYSTEM, the nonprivileged system hash table is searched; if the attribute of the control section indicates that it is a user's, then the user hash table is searched. One exception to this rule occurs when a user's authority code is P or O. In this case, the loader ignores the user hash table and searches the two system tables. Figure 42 summarizes the actions of the loader in processing the REFs and DEFs.

Notice that the loader erases the attributes of PUBLIC and READONLY from any module loaded from any library for a programmer with authority code O. The attributes PUBLIC, READONLY, SYSTEM, and PRVLGD are erased from any module loaded from JOBLIB or USERLIB for a programmer with authority code P. If a programmer with authority code P loads a module from SYSLIB, only the PUBLIC and READONLY attributes are erased.

Remember, though, that the loader does not load initial virtual storage (IVM); you will always get a public, read/write protected copy of IVM. The loader's action in assigning storage keys to control sections is governed by the attributes of those sections. (See Figure 42 and Figure 43.)

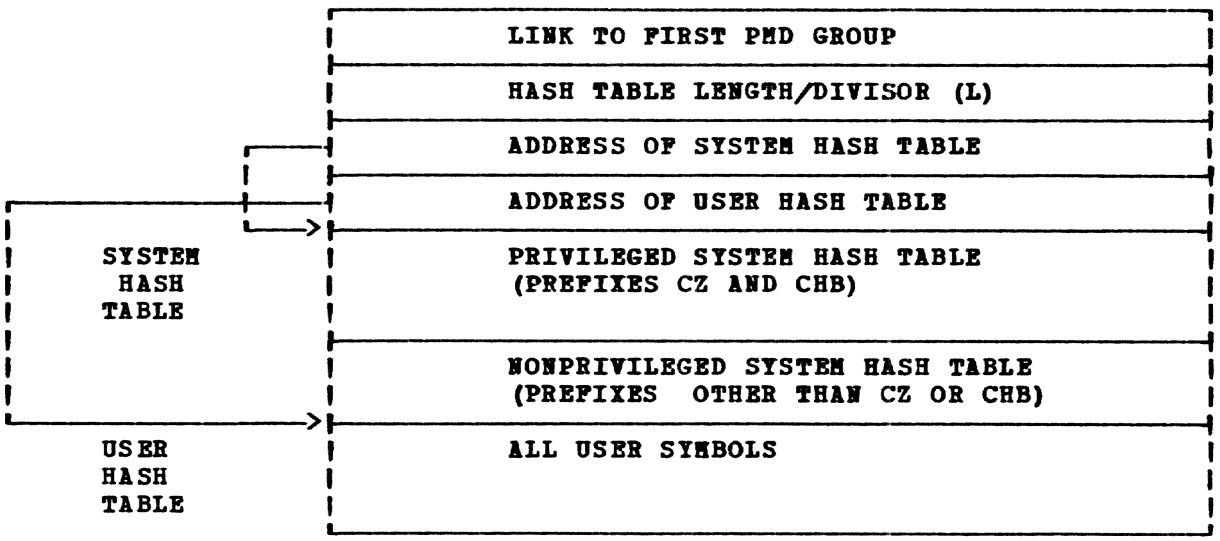


Figure 41. Dynamic loader three-part hash table

Dynamic Loader Symbol Lookup Rules

1	2	3	4	5	6
if	and	and	and	then	or
authority class is:	loader is resolving symbol from:	high-order bit of C1 or C3 byte of adcon group is:	control section containing adcon group or REF is:	lookup symbols in hash table:	If symbol is not in hash table search library:
U	Explicit LOAD/CALL or DELETE adcon group	0	SYSTEM	SYSHASHP or SYSHASHNP	SYSLIB
			NONSYSTEM	USERHASH***	ALL**
		1	NONSYSTEM	SYSHASHP or SYSHASHNP	SYSLIB
			SYSTEM	USERHASH***	ALL**
	External REF	NA*	SYSTEM	SYSHASHP or SYSHASHNP	SYSLIB
			NONSYSTEM	USERHASH***	ALL**
P or O	NA*	NA*	NA*	SYSHASHP or SYSHASHNP	ALL**

Symbol Posting Rules

1	2	3	4		5	6
if	and	then	and if		then	and
authority class is:	control section that contains DEF came from:	control section attributes may be altered:	control section that contains DEF has attributes:		DEFs may begin with only the symbols:	all legal symbols from control section are posted in:
U	SYSLIB	If control section is PRVLGD, LOADER SETS SYSTEM attribute; hence, NONSYSTEM and PRVLGD is impossible	SYSTEM	PRVLGD	CZ, CHB	SYSHASHP
				NONPRVLGD	Any but CZ, CHB	SYSHASHNP
					CZ, CHB	SYSHASHP
			NONSYSTEM		Any but SYS	USERHASH
	SYSULIB or JOBLIB	PRVLGD and SYSTEM erased.	NA*			
P	SYSLIB	PUBLIC, READ-ONLY erased.	NA*		Any	SYSHASHNP or SYSHASHP
	SYSULIB or JOBLIB	PUBLIC, READ-ONLY PRVLGD, SYSTEM erased.			Any but CZ, CHB	SYSHASHNP
O	NA*	PUBLIC, READ-ONLY erased.	NA*		Any	SYSHASHP or SYSHASHNP

*NA - not applicable, in the sense that the condition is not tested by the loader.
**ALL - the entire hierarchy of open libraries beginning at the last defined JOBLIB and ending with SYSLIB (or with that library yielding a valid definition).
***If the symbol to be resolved begins with SYS, the loader will look in SYSHASHP or SYSHASHNP, and then in SYSLIB.

Figure 42. Effect of authority code in dynamic loader processing

Class of TSS Object Module	CSECT Types For Modules	Attributes	Resultant Segment Assignment And Storage Key	
			Public	Private
SYSTEM PRIVILEGED	REENTERABLE EXECUTABLE CODE	CSECT SYSTEM, PRVLGD FXL,PUB,RDO	C	-
e.g., VAM OPEN	DATA, ADCONS, etc.	PSECT SYS,PRVLGD, FXL	-	C
SYSTEM FENCE SITTER	REENTERABLE EXECUTABLE CODE	CSECT SYS,PUB RDO,FXL	B (USER READ ONLY)	-
e.g., VAM GET	NONMODIFIABLE DATA, etc.	PSECT SYS, RDO,FXL	-	B (USER READ ONLY)
SYSTEM NONPRIVILEGED	REENTERABLE EXECUTABLE CODE	CSECT SYS,PUB RDO,FXL	B (USER READ ONLY)	-
e.g., ASSEMBLER LPC	DATA, ADCONS, etc.	PSECT SYS,FXL	-	A (USER READ WRITE)
A=Key 1 B=Key 2 C=Key 2 with fetch protection				

Figure 43. Relationship of object modules, CSECT, CSECT attributes, sharability, and storage key assignment

APPENDIX E: ORGANIZATION OF DIRECT ACCESS STORAGE

2305 DRUM STORAGE FORMAT

Each IBM 2305 drum contains 2304 pages of 4096 bytes each. Tracks 0 and 1 contain three IPL records (record one and 2 of track 0, and record one of track 1) and the IBM standard volume label (track 0, record 3). Pages 0 through 5 are not available for VAM allocation.

Dummy records of 548 bytes separate each data page to allow data channels to fetch and execute channel command words between pages. The 2305 is logically divided into 96 cylinders of 8 tracks each. Each track contains three pages and two dummy records. Figure 44 shows the organization of a typical 2305 track.

Address cchhr*	Record Size	Page Number (slot)
cchh1	4096	1
cchh2	548	dummy record
cchh3	4096	2
cchh4	548	dummy record
cchh5	4096	3

* Each track begins with IBM standard record zero. 'cc' is an integer between 00 and 95 inclusive. 'hh' is an integer between 00 and 07 inclusive.

Figure 44. Organization of an IBM 2305 Drum

DISK STORAGE FORMATS

These restrictions apply to the use of the IBM 2314 or 2311, when formatted in pages:

1. Cylinder 199 is reserved for standard error-recovery retry.
2. Page 895 (2311) is not used because of overflow restriction.

Each IBM 2314 volume contains 6496 pages of 4096 bytes. Tracks 0 and 1 contain three IPL records (records one and two of track 0 and record one of track 1) and the IBM standard volume label (track 0, record three). Pages 0 to 3 are not available for VAM allocation.

Each 2314 disk pack has 203 cylinders with 20 tracks per cylinder; each cylinder contains 32 pages. Figure 45 shows a typical organization.

Record Address CC HH R	Record Size	Page Number	Record Address CC HH R	Record Size	Page Number
*			nn 10 1	4096	16
nn 00 1	4096	0	nn 10 2	2920	17
nn 00 2	2920	1	nn 11 1	1176	17
nn 01 1	1176	1	nn 11 2	4096	18
nn 01 2	4096	2	nn 11 3	1592	19
nn 01 3	1592	3	nn 12 2	2504	19
nn 02 1	2504	3	nn 12 2	4096	20
nn 02 2	4096	4	nn 12 3	207	21
nn 02 3	207	5	nn 13 1	3889	21
nn 03 1	3889	5	nn 13 2	3136	22
nn 03 2	3136	6	nn 14 1	960	22
nn 04 1	960	6	nn 14 2	4096	23
nn 04 2	4096	7	nn 14	(1187 unused bytes)	
nn 04	(1187 unused bytes)		nn 15 1	4096	24
nn 05 1	4096	8	nn 15 2	2920	25
nn 05 2	2920	9	nn 16 1	1176	25
nn 06 1	1176	9	nn 16 2	4096	26
nn 06 2	4096	10	nn 16 3	1592	27
nn 06 3	1592	11	nn 17 1	2504	27
nn 07 1	2504	11	nn 17 2	4096	28
nn 07 2	4096	12	nn 17 3	207	29
nn 07 3	207	13	nn 18 1	3889	29
nn 08 1	3889	13	nn 18 2	3136	30
nn 08 2	3136	14	nn 19 1	960	30
nn 09 1	960	14	nn 19 2	4096	31
nn 09 2	4096	15	nn 19	(1187 unused bytes)	
nn 09	(1187 unused bytes)				

*Each track begins with the IBM standard record zero.

Figure 45. Organization of IBM 2314 volume for VAM

Each IBM 2311 volume contains 1624 pages of 4096 bytes. Tracks 0 and 1 contain three IPL records (records one and two of track 0 and record one of track 1) and the IBM standard volume label (track 0, record three). Pages 0 to 3 are not available for VAM allocation. A 2311 disk pack contains 203 cylinders of 10 tracks each; the cylinders are organized to contain 8 pages each. Figure 46 shows a typical cylinder organization.

Record Address CC HH R	Record Size	Page Number
nn 00 1	3625	0
nn 01 1	471	0
nn 01 2	3069	1
nn 02 1	1027	1
nn 02 2	2486	2
nn 03 1	1610	2
nn 03 2	1875	3
nn 04 1	2221	3
		1234 unused bytes, track 4
nn 05 1	3625	4
nn 06 1	471	4
nn 06 2	3069	5
nn 07 1	1027	5
nn 07 2	2486	6
nn 08 1	1610	6
nn 08 2	1875	7
nn 09 1	2221	7
		1234 unused bytes, track 9

Figure 46. Format of IBM 2311 volume for VAM

Each IBM 3330 model 1 contains 23,427 pages of 4096 bytes. Dummy records of 102 bytes separate each page on a track. Tracks 0 and 1 con-

tain three IPL records and the IBM standard volume label. Pages 0 to 6 and 23,423 to 23,426 are not available for VAM allocation.

Each 3330 model 1 disk pack has 411 cylinders with 19 tracks per cylinder. Each track contains 3 pages. Figure 47 shows a typical track layout.

Address cchhr*	Record Size	Page Number (slot)
cchh1	4096	1
cchh2	548	dummy record
cchh3	4096	2
cchh4	548	dummy record
cchh5	4096	3

* Each track begins with IBM standard record zero. 'cc' is an integer between 00 and 410 inclusive. 'hh' is an integer between 00 and 18 inclusive.

Figure 47. Organization of an IBM 3330 Disk

Each IBM 3330 model 11 contains 46,455 pages of 4096 bytes. Pages 0 to 6 and 46,451 to 46,454 are not available for VAM allocation.

The 3330 model 11 is identical to the 3330 model 1 in track capacity and tracks per cylinder. However, the 3330 model 11 has 815 cylinders compared to 411 for the 3330 model 1.

Each IBM 3350 contains 65,400 pages of 4096 bytes. Dummy records of 525 bytes separate each page on a track. Tracks 0 and 1 contain three IPL records and the IBM standard volume label. Pages 0 to 7 and 65,395 to 65,399 are not available for VAM allocation. Each IBM 3350 has 560 cylinders with 30 tracks per cylinder. Each track contains 4 pages. Because of internal system restrictions only 529 cylinders are formatted and used by TSS. Figure 48 shows a typical track layout.

Address cchhr*	Record Size	Page Number (slot)
cchh1	4096	1
cchh2	525	dummy record
cchh3	4096	2
cchh4	525	dummy record
cchh5	4096	3
cchh6	525	dummy record
cchh7	4096	4

* Each track begins with IBM standard record zero. 'cc' is an integer between 00 and 528 inclusive. 'hh' is an integer between 00 and 29 inclusive.

Figure 48. Organization of an IBM 3350 Disk

All RTAMLOG entries are 32 bytes in length. The first 16 bytes are a common header which contains the module and entry IDs, the SDA or RID of the device, and 8 bytes of common information. The second 16 bytes are entry dependent.

Module IDs are set up as follows:

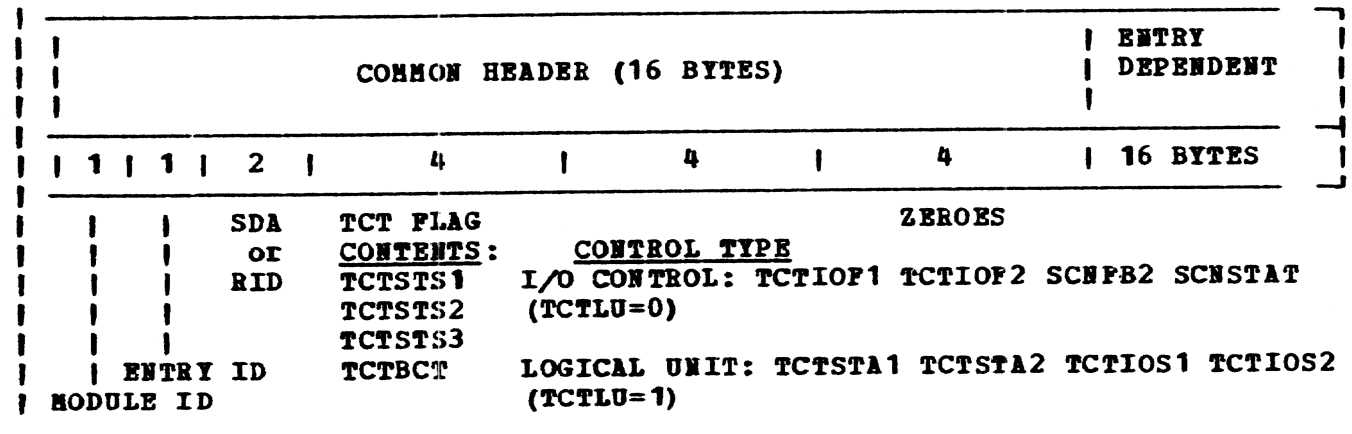
X'00' - RTAM services entry
X'01' - CEDM01
X'02' - CEDM02
X'03' - CEDM03
X'04' - CEDM04
X'07' - CEDM07
X'08' - CEDM08
X'09' - CEDM09
X'0A' - CEDSSCP
X'0B' - CEDM0B
X'0C' - CED37XX
X'0D' - CEDLUCP
X'0D11' to X'0D16' - CED327R
X'37' - CEDBM

The 8 bytes of common information are arranged as follows:

the first 4 bytes are common to all entries;

the second 4 bytes are dependent on the control type which is determined by the TCTLU bit in the TCT; if this bit is a zero the control type is called an 'I/O control type', and if the bit is a 1 the control type is called a 'logical unit type'.

The format of the RTAMLOG entries are as shown below:



APPENDIX G: USER LIMITS TABLE

When a user is joined to the system, he is assigned a set of limits which determine maximum amounts of system resources he may use. Which set he is assigned is specified in the RATION operand of the JOIN command. Two sets of limits (whose values are listed in the figure below) are supplied with the system; the first set (key 1) represents limits for the system programmer, the second set (key 2) represents limits for the privilege class D user. The installation may modify these values or add additional sets. The table containing these sets is called the user limits table; each set of limits is an entry in the table. An entry represents a record in SYSULT, a VISAM member of the VPAM data set SYSLIB. The layout of an entry in the user limits table is defined in the DSECT CHAULT.

The pre-joined users named TSS, SYSMANGR, and SYSOPER0, are given a set of limits independent of the user limits table. These values are also shown in Figure 51.

Item	Set 1	Set 2	Set...	TSS, SYSMANGR, SYSOPER0 [3]
Limits Category (Key)	1	2		-
CPU Time (milliseconds)	2,000,000[1]	2,000,000[1]	Installation	X'7FFFFFFF'[4]
Connect Time (seconds)	2,000,000[2]	2,000,000[2]	may modify	X'7FFFFFFF'[5]
Task Count	200	20	Sets 1 and 2	200[6]
Aux. Storage Pages	2,000	500	or add up to	400
Temporary Public Pages	20,000	2,000	7 more sets	20,000
Permanent Public Pages	20,000	1,000	(7 more	20,000
Direct Access Devices	20	1	entries in	20
Magnetic Tape Drives	20	1	User Limits	20
High-Speed Printers	20	0	Table).	20
High-Speed Readers/Punches	20	0		20

[1] 2,000,000ms = 0 hrs:33min:20sec.
[2] 2,000,000 secs = 555 hrs:33min:20sec.
[3] Independent of User Limits Table; shown for comparison.
[4] X'7FFFFFFF' ms = 1,118 hrs:28min:51secs.
[5] X'7FFFFFFF' secs = 1,118,481 hrs:03mins:59secs.
[6] 400 FOR USER NAMED TSS.

Figure 51. System-supplied values for user limits table

APPENDIX H: FACILITIES BY PRIVILEGE CLASS AND AUTHORITY CODE

Individuals assigned particular privilege classes and authority codes may employ facilities available only to their class or code. A summary of the facilities associated with each class or code follows.

Privilege Class A - System Operator
ASNBD, BCST, CANCEL, DROP, FORCE, HOLD, MSG, PRINT, REPLY, RT, SHUTDOWN, and USAGE command options indicated in Operator's Guide; all commands in Command System User's Guide (not requiring additional privilege).

Privilege Class B - System Administrator
Same as class F except that the privilege class B user can join user IDs of only one-to-six characters, prefixed by the system with first two characters of administrator's ID, and he can only assign D privilege classes.

Privilege Class D - User
All commands in Command System User's Guide requiring no further privilege or authority (for example, UPDTUSER requires an O authority code).

Privilege Class E - System Monitor
Use of MSAM and TAMII macro instructions; additional BSAM and QSAM options; On Line Test System (OLTS) facilities; Virtual Memory Error Recording Procedures (VMEREP); and ability to refer to devices symbolically.

Privilege Class F - System Manager
CANCEL, JOIN (full eight-character user IDs and all privilege classes except A and F), LOGON, LOGOFF, DSS?, LINE?, and QUIT command options indicated in Manager's and Administrator's Guide: all commands in Command System User's Guide (not requiring specific authority or privilege codes); all command facility options available to B class user.

Privilege Class G - Special
Can DDEF restricted DASD UNITS.

Privilege Class T - MTT Administrator
Can create an MTT task.

Authority Code U - User

1. Cannot invoke Time Sharing Support System (TSSS), but can use it if connected by a master system programmer.

2. PCS Usage

   - Can use PCS to display (using absolute virtual addresses) nonprivileged system routines residing in virtual storage.

   - Can use the PCS DISPLAY and SET (using symbolic labels) commands in his own public or private nonprivileged virtual CSECTs and can use AT in private nonprivileged virtual CSECTS.

   - Cannot use PCS to display privileged CSECTs, public system CSECTs, or to display (symbolically) areas of nonprivileged system CSECTs, because system symbols will not be resolved.

3. Can code and assemble or compile nonprivileged or privileged code (obtaining privileged macro expansions by use of the DCLASS macro instruction), but cannot load or execute any privileged code (as privileged) from his USERLIB or job libraries because of dynamic loader protection. U authorization programmers who assemble privileged CSECTs must have O authorization programmers load and execute that code as privileged.

Note: The dynamic loader erases the PRVLGD and SYSTEM attributes to prevent U authorization programmers from changing privileged system code. U authorization programmers can run privileged system code as nonprivileged by creating their own IVM symbols and overlaying special SVC instructions.

4. Could have O authorization programmer put a privileged CSECT he had created into SYSLIB and could then execute that code (depending on whether that code refers to system data sets).

5. Can execute any privileged code in SYSLIB from nonprivileged code only indirectly by establishing the proper linkage.

Authority Code P - Nonprivileged System Programmer

1. Can use TSSS as a master system programmer (MSP) or as a task system programmer (TSP) if residing at the terminal.

2. PCS Usage

   - Can use PCS to DISPLAY from public or privileged CSECTs; cannot SET or AT in public or privileged code.

   - Can use PCS to DISPLAY (symbolically), SET, or AT into private nonprivileged system code as well as into his own nonprivileged routines (loaded from his USERLIB and JOBLIBs) that reside in virtual storage.

   - Because the dynamic loader strips the PRIVILEGED attribute from privileged system modules when they are loaded for a P authorization programmer (from USERLIB or JOBLIB), the private copy he receives is nonprivileged, and he can debug and alter that copy.

3. Can assemble or compile nonprivileged or privileged code (obtaining privileged macro expansions by use of the DCLASS macro instruction) but cannot load this privileged code (as privileged) from his USERLIB or JOBLIB because of dynamic loader protection.Therefore he will not be able to execute privileged code (as privileged) from those libraries.

4. He can execute any privileged code in SYSLIB from nonprivileged code by dynamically loading the code and establishing appropriate linkage.

Authority Code O - Privileged System Programmer

1. Can use TSSS as an MSP or TSP if using the proper terminal.

2. PCS Usage

   - Can use PCS to DISPLAY or SET in public or privileged system code.

   - Can use PCS to DISPLAY, AT, or SET in nonprivileged virtual storage.

3. Can code nonprivileged or privileged code (in conjunction with DCLASS) to go into USERLIB, SYSLIB, or a job library and can execute privileged code from any of these libraries. He can also execute any privileged code or write into any privileged code.

4. Has exclusive right to the LVPSW macro instruction (SVC).

5. He alone can execute privileged code.

6. Can DISPLAY, SET, or DUMP from IVM.

7. Use of CVV, UPDTUSER facilities.

8. Can open privileged system data sets.

PROBLEM / EXAMINE	SYSERR[1] SUPERVISOR	SYSERR[1] USER TASK	PAGING FAILURE[2]	WAIT STATE[3]	LOOPS[4]	COMMENTS
CEAJIL Supervisor Interruption Log	X	X	X	X	X	Starts at entry point CEAJIL. (See Fig. 53)
CZCJTL Task Monitor Interruption Log		X			X	Starts at entry point CZCJTL. (See Fig. 54)
CHBDAI* Direct Access Interface Block			X			Points to Page Error Control Control Block.
CHBISA* Interrupt Storage Area		X				Segment 0, Page 0. X'6C0'=VPSWs and regs 13-4. X'6B8'=old VPSW for last task interruption. X'7D0'=old program VPSW. X'880'=start of last IORCB passed back from Supervisor
CHBSCN* Scan Table			X	X		Entries for system device and interruption queues.
CHBSMC* Scan Master			X	X		Provides data to Queue Scanner for search of Scan Table.
CHBSYS* System Table			X	X		Contains TSI chain pointers and system parameters.
CHBTSI* Task Status Index			X	X		Contains information used by Supervisor for task execution
CEAIS[1] SYSERR Save Area	X	X				Contain forward and backward links. Contain general registers at the time the module made a call. (See Fig. 55)
CEEAQB Paging Failure Save Area			X	X		

*See System Control Blocks for DSECT description, map, and listing.

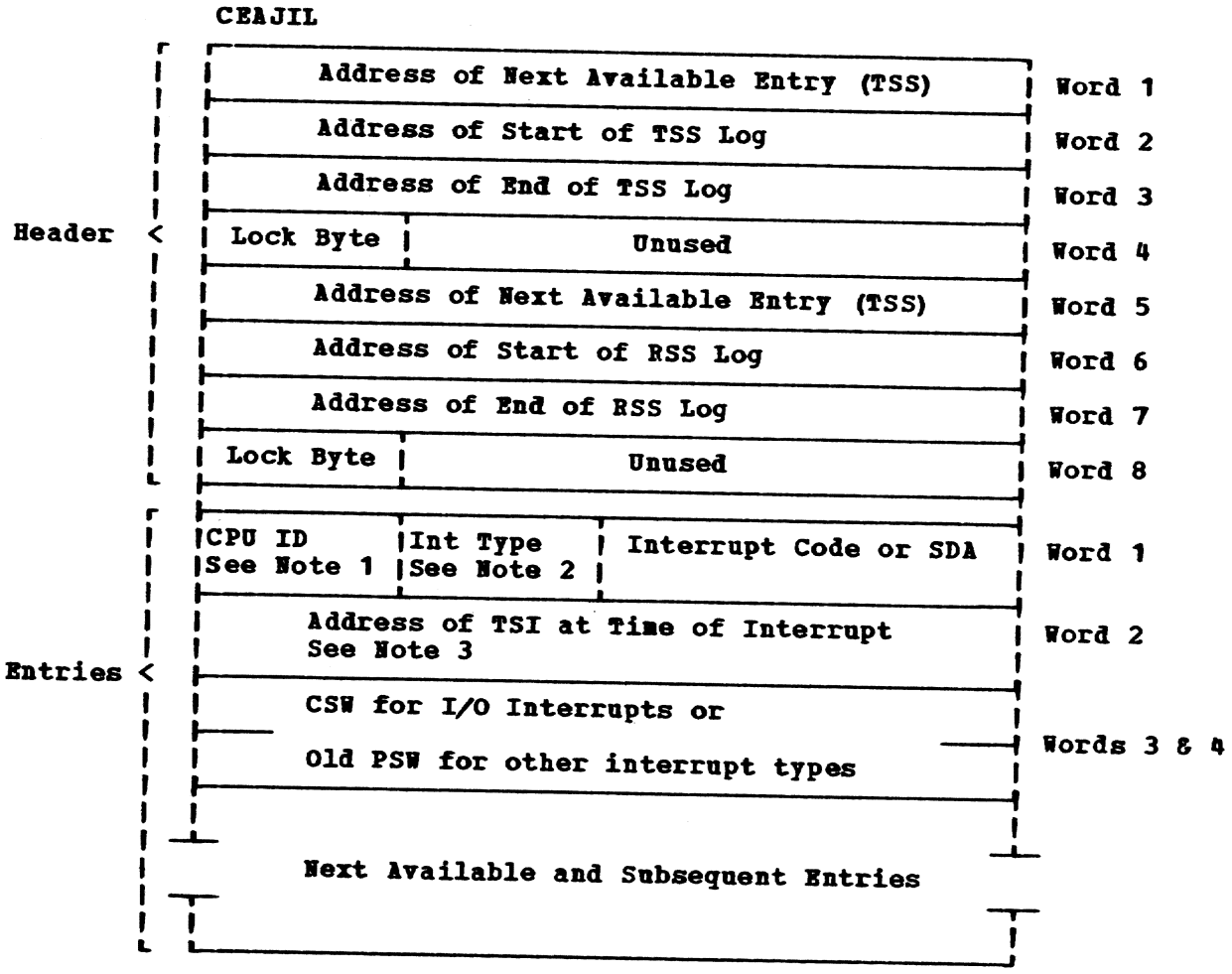
[1]All SYSERRs use the save area in CEAIS. A Supervisor-issued SYSERR may be the result of a VM problem, so the SYSERR meaning will determine which type of dump must be taken. If the SYSERR is not listed in current documentation, check the interruption log. If last SVC 254 found with relocate bit in PSW (bit 5) off, Supervisor issued SYSERR. If last SVC 228 found with relocate bit on, SYSERR was issued from task. Address in PSW indicates where SYSERR was issued.

[2]The Paging Failure Recovery save area is filled after a solid paging failure has occurred. On intermittent paging errors that cause SYSERRs, VMREPs are useful. Check the Direct Access Interface Block (CHBDAI) for a pointer to the Paging Error Control Block (CHBPEC) for the failing I/O operation.

[3]Check status of tasks and work known to be in the system:
Check TSI of current task for 'delay' or 'ready' status. GQEs may be queued on task.
Check Scan Table for active work and suppress conditions. Examine pointers to GQEs queued on Scan Table. (GQEs may also be queued on shared External Page Table entries.)
Check Supervisor Interruption Log for I/O interruptions indicating error conditions.
(Possiblilities: AWAIT SVC issued without preceding IOCAL SVC; relocation interruption which may have led to page wait condition.)

[4]Check register D, using roller 1 in position 2. Current PSW is on roller 4, position 1. Instruction step CPU to verify loop. If relocate bit (bit 5, current PSW) is on, loop is in task. If relocate bit is off, loop is in supervisor. If loop encompasses both supervisor and task, instruction stepping may take too long. Find point of transfer between supervisor and task by running in normal mode, but stopping on the address of the XTSI.

Figure 52. Data areas to examine for common system problems



Note 1: CPU 1 = 80
CPU 2 = 40

Note 2: External = 18
SVC = 20
Program = 28
I/O = 38

Note 3: For an I/O Interrupt in Supervisor state

Word 2 = Byte 1 of Ext PSW
Bytes 2-4: Instruction Address

For RSS Program and I/O interrupts:

Word 2 = A pointer to TSS LOG where the interrupt would have been recorded.

Figure 53. Supervisor interruption log

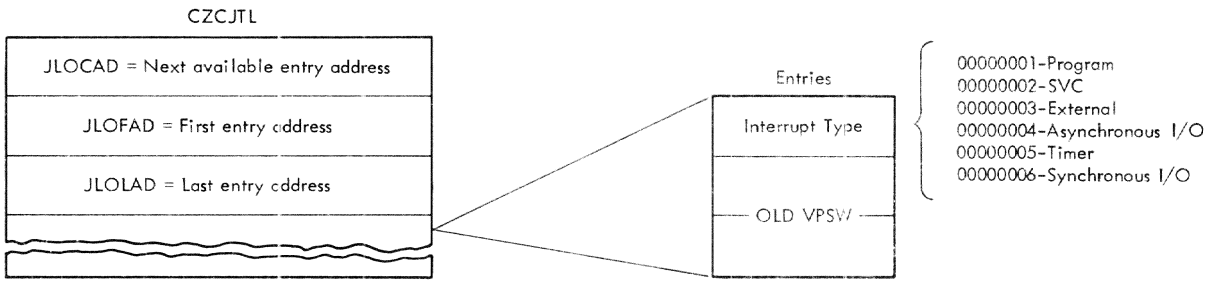


Figure 54. Task monitor interruption log

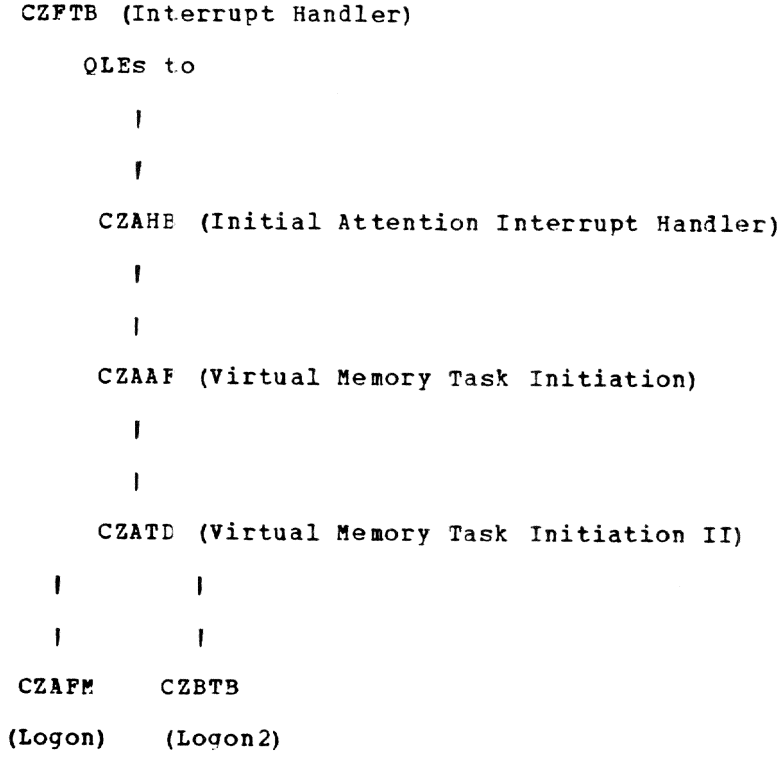
-Contains the length of this save area	WORD 1*
-Contains a pointer to the save area of the CALLING program (chain backward)	WORD 2
-Contains a pointer to the save area of the CALLED program (chain forward)	WORD 3
-Contains the return linkage	WORD 4
-Contains the entry point address	WORD 5
-Contain the contents of registers 0 - 12	WORDS 6 - 18
-Contains the address of the CALLING program's copy of the PSECT for the CALLED program	WORD 19*
*CALLING program's responsibility	

Figure 55. Save area format

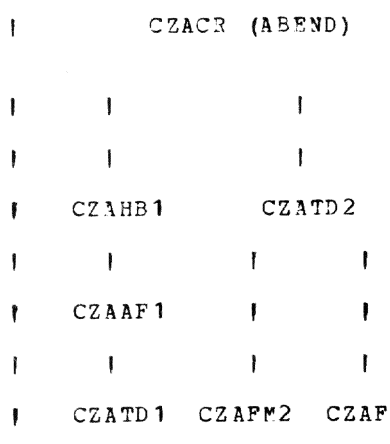
APPENDIX J: LOGON PROCESSING OVERVIEW

The changes to LOGON for PRPQ2 are basically in the handling of required datasets, especially for SYSOPER0 and pod owners. Every attempt is made to complete startup even with back level datasets or without certain datasets. This was not designed to recover from all system problems but to give the system programmer a chance to fix datasets and then to re-ipl or continue. Note that most cases will require a re-ipl. Below is an overview of the logic flow at logon time both for initial logon and as a result of abend. This remains the same. After the diagrams is a detailed logic flow of the key modules in the LOGON process, updated for the new catalog structure, for this new LOGON code and for all other PRPQ2 and pre-PRPQ2 functions. In addition, attached is a module-by-module description of what happens at each particular dataset failure and the attempted recovery procedures.

Overview of the LOGON Process



AFTER AN ABEND



Functions of Modules in the LOGON Process

CZAHB1

- Mark task conversational (ISACOV,TCMCOV)
- XTRCT userid and move to TCMUID
- XTRCT taskid
- If not taskid 1 turn off operator flag (TCMOP)
- Use taskid to search pool table (Call CZCBA1)
- If userid not usable (UIDRSV,UIDMOVA,UIDMOVS) or delete pool in progress (APIDEL), do TFREE and DLTSI
- If the pool is in maintenance, allow only prejoined user on (APIUSR) and set up as maintenance taskid (APIMTID)
- Setup poolid (TCMPID)
- Set up PVT pointer in TCMPVT (from APIPVT)
- Call CZAAF1
- Return

CZAAF1

- Move ftn interrupt recovery flg (TCMFIR from SCMFIR)
- Reserve TDY space (RSVSEG)
- Disconnect TDY (DISCSEG) and turn off TDY connected flag (ISATDC)
- For taskid 1 only, add system pool devices to TSDL using the public volume table (from APIPVT)
- Search the API for users poolid (using TCMPID for compare)
- Move fields from API to SYSSVCT JFCB (TDT028,prebuilt)

```
|     OPEN SYSSVCT (CZCFX5)
|
|     For taskid 1 only, go through SYSSVCT and build userid table
|
|       If join is in progress, no entry is constructed (UCTJIP)
|       Fields referenced or changed include (UCTCON*,UIDNCAT,
|         UIDCOPY, and the sequence number UIDNSEQ)
|       Fields moved include 'reserved for maintenance' bit,
|         'move user started' bit, PVN limits
|       (GETMAIN, SETL B, GET, construct entry)
|       Disconnect userid table (call CZCBA2) and then
|         FREEMAIN the temporary table space.
|     For taskid 1 only, call CZCFW4 to create the dataset
|         'system pool owner.SYSCAT2'
|       Call CZCFL2 to locate all TBLOCKS with that dsname beginning
|       If non zero return code, FREEMAIN all TBLOCKS and continue
|       If zero, loop through each TBLOCK and call CZCBA3 to take
|         the DSCB and put it into the uid table (CHBUID)
|       When done, call CZCFP1 to flush the old SYSCAT, call CZCFW1
|         to build a SYSCAT2 for the poolowner, FREEMAIN the TBLOCKS
|         and continue.
|     Load address of USERCAT JFCB (TDTCAT, prebuilt) and mark
|       privileged access (TDTVPY) and catalog sensitive (TDTRCT)
|     DDEF SYSUSE
|     DDEF SYSLIB
|     DDEF SYSPLIB
|     DDEF SYSRCS
|     DDEF SYSUSE
|     DDEF SYSMAC
|     DDEF MACNDX
|     Call CZCFW1 to build the 'poolowner.SYSCAT2.userid dataset'
|     If not the oper./BIO task(TID=1/2) call timer routine (CZAVB1)
|     Call CZATD1
|     Return
```

CZATD1

Open SYSLIB (CZCTJ8)
FIND SYSMLF member
If not a bio task(NTCBID=0) and not a conversational task (TCMOV)
  RELEASE the SYSIN JFCB
  DDEF sysin dataset (DCB is CZAHC6)
  turn off conversational flag in NTC (NTCSW4)
  call CZBTB4
If a bio task, skip rest and return
If a conversational task,set flag in NTC (NTCSW4) and in PSECT
XTRCT TASKID and put in TCM and NTC
If operator task
  OPEN SYSUSE (DCB in task common)
  (GET, reset fields, WRITE record back)
  CLOSE SYSUSE
If abend is creating this task, return (ISAABN)
TGATRD (Read first sysin record)
Look for LOGON
Check userid for valid characters
If taskid is 1 call CZAFM2; if not, call CZAFM1
Set conversational flag in NTC
Basr to CZBTB1 (LOGON2)
If express batch task, call CZCUA to do RCR OPEN for SYSOPER0
Then if task is 1
  Call CZACB2
  Turn on task initiated flag (ISATI)
  Turn off logon in progress (ISALP)
  Return
Or if task is a bio task (NTCBIO) (ex. RT,WT)
  Turn on task initiated (ISATI)
  Turn off logon in progress (ISALP)
  Return

```
        If bio type task (BIO or NETWORK)

          Turn on task initiated flag (ISATI)
          Turn off logon in progress (ISALP)
          OBEY 'ZLOGON'
          If abend is creating the task, just return (ISAABN)
          If not, call CZCJQS (QLF to C.A.  & E.)
        If none of the above (simple conv.  task)
          If not a pristine logon (NTCTSK) and not created by abend(ISAABN)
            FINDDS for SYSMAIL dataset
            If one exists RELEASE the DDEF and inform user he has mail
          Turn on the task initiated flag (ISATI)
          Turn off the logon in progress flag (ISALP)
          OBEY 'ZLOGON'
          If the task is being created by abend (ISAABN), just return
          If not, call CZCJQS (QLF to C.A.  & E.)

      CZATD2
        If batch monitor or network task or abend creating
          a new task, call CZAFM2
        In other cases, call CZAFM4
        If not abend creating a new task, return
        If abend is creating a new task (ISAABN) then
          branch into CZATD1 code at BASR to CZBTB1

 CZAFM2                          CZAFM1
```

Note: the flow for CZAFM2 consists of only those items listed directly below CZAFM2. The flow through CZAFM1 consists of all the items in CZAFM2 plus those items listed directly under CZAFM1 (all lines with an asterisk).

```
 If shutdown in progress(SCMITI),abend
 Call CZUFM1 to process keywords
                                 *If not abended task(ISAABN) or bulkio type
                                 *(TCMBIO) or SYSOUT OPEN, PRMPT CZAFM000
 Search JFCB chain for SYSUSE JFCB
```

```
| Mark as catalog sensitive, privileged in TDT

| Set super privileged bit in ISA
| REDTIM and save in TCMTOS
| Search the APE to find the pool owner's userid
| OPEN SYSUSE
| For express batch job, get the userid from the logon card
|     and SETUP in TSI
| XTRCT and SETUP conv flag in TSI
| Call CZCUA (RCR OPEN) if SYSUSE is open (TCMFP3)
| Call CZCUA (RCR OPEN) for TSS and poolowner userid if
|   not already done
| IF conversational (TCMCOV)
|   Call CZCUA R (CPU time)
|   Call CZCUA (CONN time)
| If non-conversational
|   Call CZBTBA to DDEF and OPEN SYSOUT
|   GATWR-write out LOGON card
|   If not oper/bulkio type task (TCMOP,TCMBIO)
|     PRMPT CZAFM000
|   Call CZCUA R
|   Call CZCUA type=TASK
|   Set non-conv. print flag on (TCMPNT)
| User privilege to TCM
| Set user table updated (TCMABS)
| Update user table entry (WRITE)
| If shutdown in progress (SCMITI), abend
|                                 *Process password
| Set logon ok (TCMLOK)
| Process addressing and charge number parameters
| Set up privilege class of user (TCMPRV)
| Turn off confirmation flag (TCMCOF)
| Set full message indicator (TCMOPT)
```

```
| Process CSECT backing, aux., pristine and xivm parameters

| If non/conversational task and not a special
|   task or call to CZAFM4, SIP for timer interrupts (TCMTIME)
| SETUP user priority
| SCHED
| SETUP authorization
| Move authorization to ISA
| If USEAIH=0 or P
|   MAPTDY T (connect TDY)
|   Set up for system hash
|   MAPTDY U (connect user)
| If pristine option is not X, DDEF USERLIB
| Get time and date of logon (EBCDTIME)
| Logon time to TATON
| If non/conversational, BSN to TATMID and
|   BSN to AULTMID
| Set up salutation messages
|                                *If not abended task, PRMPT salutation message
|                                *WTL message
|                                *AUXSET
| If abended task, WTL and AUXSET
| Return

|      CZBTB1

|        XTRCT taskid

|        Set 0 default number in NTC (NTCDNO)
|        Set attention indic.  character (NTCAIC='5a')
|        Call CZBTB2
|        Call CZBTB3
|        If a conversational task
|          Mark dummy SYSIN DCB (SYSINDCB) as open (DCB03)
|          Put SDAT entry addr for SYSIN term.  in TCM (TCMSIN)
```

```
      SIR (Set attentions to go to CZASB1)

      If RTAM-do SETTCT
    Return

  CZBTB2
    If not a pristine logon-OPEN USERLIB(SYSPRD)
                              OPEN USERLIB(SYSPRO) (DCB=CZATE8)
                              OPEN USERLIB(SYSMLF) (DCB=CZATJ9)
                              FIND USERLIB(SYSPRO)
                                If not found, CLOSE (CZATE8)
                              Reset DCBLRE fields to 256
                              FIND USERLIB(SYSMLF)
    If not an express logon-OPEN SYSLIB(SYSPRO)
                              FIND SYSLIB(SYSPRO)
                              OPEN SYSLIB(SYSPRD)
    Return
  CZBTB3
    GETMAIN 1 page for input buffer
    Call CZASD2 (STARTVAR)
    If not a pristine logon and USERLIB(SYSPRO) exists
      FIND USERLIB(SYSPRD)
        If not found (rc=20)
          Get addr of DCB for SYSULIB(CZBTB8)
          SETL B
        If not found (rc=4)
          SETL B (DCB=CZATE8)
          GET next record
          Call CZBTB9 (proclib scan)
          Validate and process record
          HASH
          Call CZASD3
          Call CZASD4/CZASD5 to search chain again or
            to insert current entry
```

```
|           On EODAD, go to CZBTB5

|         If found (rc=0)
|           Read the dictionary
|           Call CZBTBX
|       If SYSLIB(SYSPPO) exists
|         FIND SYSLIB(SYSPRD)
|         SETL B
|         Call CZBTBX
|       If not a pristine task
|         FIND USERLIB(SYSPRX)
|         If found
|           Get address USERLIB dcb (CZBTB8)
|           GETMAIN
|           Call CZBTBX
|           If not pristine logon
|             FIND userlib(SYSPCL)
|             If found, TERMPRO ACTION=R...
|             If not found, MCAST,TERMPPO ACTION=W
|           Close SYSLIB dcb (CZBTB7)
|           Close dcb (CZBTB8)
|           Return
|       If a pristine task or USERLIB(SYSPRX) not found
|         FIND SYSLIB(SYSPRX)
|         If not found, issue msg, close dcbs and exit
|         Get addr of syslib dcb (CZBTB7)
|         Go to GETMAIN,call CZBTBX and then close (CZBTB7,CZBTB8)
|         Return

|   CZBTB4
|       FINDJFCB for SYSIN ddname
|       Set sysin sda in TCM (TCMSIN from TDTID1)
|       SETUP sysin
|       Flag as TSS SYIN/SYSOUT (NIBTSS
```

OPNDST (call TAMII to connect) (CZFTF1)

Set format of record (TCMGRD to 0 for variable
and 1 for fixed)
Return

CZBTBA
RELEASE SYSOUT
DDEF SYSOUT
FINDJFCB for SYSOUT ddname
Set SYSOUT sda in TCM (TCMSOT from TDTID1)
SETUP sysout
Flag as TSS SYSIN/SYSOUT (NIBTSS)
OPNDST (TAM II connect) (CZFTF1)
Set record length in TCM (TCMLNG to 132)
Return

During LOGON processing several datasets are ddeffed
and opened. If any of these ddefs or opens fails, the task
SYSERRS and-or ABENDS. If the user happens to be SYSOPER0,
the system will not startup.

Design Solution

For users and SYSOPER0, a USERLIB DDEF or OPEN failure will
result in a message and a pristine LOGON. In addition, for
SYSOPER0 or a conversational user, the LOGON processors
will attempt to open older generation datasets where applicable
and to temporarily bypass other failures for SYSOPER0.
In the SYSOPER0 case, since 'OK' has not been entered, the
system operator or system programmers can correct and/or
re-ipl. To aid in problem determination, the messages will
contain the dataset name, what problem was found and how
it was temporarily bypassed. Every effort will be made to
complete startup.

Module Impacts

CZAAF (VMTI)

Set up special abend recovery fields in task common for TID 1
and the pool owners only
On ABEND on OPEN of SYSSVCT
TGATWR message, set up abend recovery again but skip the
the build of the userid table
DO NOT RUN AFTER FIXING PROBLEM, RE-IPL.
On ABEND on SETL B or GET OF SYSSVCT dataset
TGATWR message, setup ABEND recovery again and branch to the
EODAD routine.
DO NOT RUN AFTER FIXING PROBLEM, RE-IPL.
On ABEND during 'copycat' processing (for system pool)
TGATWR message, setup ABEND recovery again and
skip to USERCAT processing.
DO NOT RUN AFTER FIXING PROBLEM, RE-IPL.
Generally, where a major SYSER would have been issued, if
the TID is 1, or the userid is the pool owner, these
have been converted to minors and code added to
continue the LOGON process
added to continue the LOGON process.
In addition, non-zero return codes from DDEF are processed as
For SYSLIB(0) - Issue message via TGATWR
Try to DDEF SYSLIB(-1), if successful,
continue
If not, ABEND 2 if the TID is
not 1 or the pool owner; if TID is
1 or the pool owner issue minor
SYSER 1,2,5,1,61,02 and continue
For SYSUSE - Issue message via TGATWR
Abend unless TID is 1 or pool owner, then

```
                        issue SYSER 1,2,5,1,61,9 and continue
        For SYSMAC/MACNDX - Issue message and attempt to DDEF at the
                        (-1) level; If unsuccessful, issue
                        SYSER 1,2,51,61,10 and continue
        For SYSPLIB - try to DDEF SYSPLIB(-1)
                        If successful put out message
                        If unsuccessful continue
        For SYSRCS - continue with no message
    NOTE:  As the datasets are DDEFFed and/or OPENed, flags
      indicating the status are set in the logon footprint fields
      in task common.  They are checked by later modules that
      wish to access or further process those dataset.

    CZATD

    Set up abend recovery fields in task common for TID 1
      and pool owners only
    If the task common footprint indicates that SYSLIB is DDFFed
      On ABEND on OPEN of SYSLIB(SYSMLF)
        Issue a message via TGATWR, reset ABEND field in task common
          and skip the FIND of SYSLIB(SYSMLF) and continue
      On ABEND on FIND of SYSLIB(SYSMLF)
        Issue a message via TGATWR, reset ABEND field in task common
        'CLOSE' the DCB
        Reset the SYSLIB(SYSMLF) 'open' flag in footprints
        Set up ABEND fields in task common and continue

    CZAFM (LOGON)

    Search for the JFCB for SYSUSE
      If not found and it is supposed to be DDEFed (footprint)
        Issue a minor SYSER (1,20,55,62,11) and ABEND
      If not DDEFed and NOT the operator task or pool owner
        a minor SYSER is issued (as above) and the task is abended.
```

```
|       If not DDEFed and the task is the operator task, continue
|
|     For taskid 1 or pool owners only, set up special abend flags in
|       task common
|     If SYSUSE is supposed to be DDEFed
|       On OPEN of SYSUSE failure (for SYSOPER0 or pool owners only)
|         Set up a dummy SYSUSE entry in CZAFMs PSECT and continue
|     If SYSUSE is not open, skip all RCR processing
```

INDEX

Where more than one page reference is given, the major reference is first. All references are within plus or minus one of the indicated page number.

This Newsletter No. GN20-4106
Date: July 1, 1980

Base Publication No. GC28-2008-5
File No. S370-36

Previous Newsletters None

IBM Time Sharing System
System Programmer's Guide

This Technical Newsletter, a part of PRPQ 5799-AYX and PTF 3.12 for the IBM Time Sharing System, provides replacement pages for the above-mentioned publication. Pages to be replaced (or added) are:

Title-Abstract	160.1 - blank (add)	239 - 240
v - x	171 - 172	240.1 - 240.2 (add)
39 - 40	172.1 - 172.2 (add)	242.3 - 242.7
40.1 - 40.3 (add)	177 - 178	243 - 244
56.1 - 56.4	178.1 - 178.2 (add)	265 - 268
59 - 60	187 - 188	271 - 272
60.1 - blank (add)	188.1 - blank (add)	285 - 286
78.1 - 78.2	238.1 - 238.4	286.1 - 286.14 (add)
157 - 160		287 - 294

Summary of the changes:

- Macros and commands have been updated to reflect new operands, new examples of use, etc. Also, editorial corrections have been made.
- Complete descriptions of the following commands/macros have been added:

  FIXDSCB QSVC ROPAGE

Changed areas in text and figures are indicated by a vertical bar in the left and/or right margins.

Please file this cover letter at the back of the manual to maintain a complete record of the changes.

IBM Corporation, Time Sharing System, Dept 80M, 1133 Westchester Avenue,
White Plains, New York 10604

Printed in U. S. A.