GC28-0628-1

Preface

This publication describes supervisor facilities that can be influenced by the system programmer.

Part I: Supervisor Services discusses supervisor functions restricted to system programmers and installation-approved personnel, and the macro instructions and parameters used to obtain the functions.

Part I is divided into seven topics. For convenience and compatability, these services are grouped in the same manner as in **OS/VS2 Supervisor Services and Macro Instructions**, GC28-0683. Accordingly, some of the subjects discussed are merely continuations of previous coverage, but are extended to include discussions of the new functions. However, some subjects discussed under the topics are completely new, but are restricted in use to the system programmer.

Part II: Macro Instructions contains the formats and descriptions of the supervisor macro instructions. It provides system programmers with the information necessary to code the macro instructions.

Publications referenced:

OS/VS2 System Programming Library: Data Management, GC26-3830.
IBM System/370 Principles of Operation, GA22-7000.
OS/VS2 Supervisor Services and Macro Instructions, GC28-0683.
OS/VS2 Scheduler and Supervisor Logic, SY28-0624,SY28-0625,SY28-0626. (3 volumes)
OS/VS - DOS/VS - VM/370 Assembler Language, GC33-4010.
OS/VS2 System Programming Library: System Generation Reference, GC26-3792.
OS/VS2 System Programming Library: Debugging Handbook, GB0F-8211.
OS/VS2 System Programming Library: Initialization and Tuning Guide, GC28-0681.

Contents

Figures

Part I: Supervisor Services

The supervisor provides the resources that your programs need while assuring that as many of these resources as possible are being used at a given time. Well designed programs use system resources efficiently. Knowing the conventions and characteristics of the VS supervisor will help you design more efficient programs.

This section describes those supervisor services that should be restricted in use to systems programmers and installation-approved personnel. In most cases, the services correspond to macro instructions and parameters that are described in part II.

For convenience and compatibility, the services you can request from the supervisor are grouped in the same manner as in **OS/VS2 Supervisor Services and Macro Instructions**. The service groupings may be described as follows:

Subtask Creation and Control: Occasionally, you can have your program executed faster and more efficiently by dividing parts of it into subtasks that compete with each other and with other tasks for execution time.

Program Management: The supervisor can be used to aid communication between segments of a program. Save areas, addressability, and passage of control from one segment of a program to another are included in this topic.

Resource Control: Portions of some tasks depend on the completion of events in other tasks, thus requiring planned task synchronization. Planning is also required when more than one program uses a serially reusable resource.

Interruption, Termination, and Dumping Services: The supervisor provides facilities for writing exit routines to handle specific types of interruptions. It is not likely, however, that you will be able to write routines to handle all types of abnormal conditions. The supervisor therefore provides for termination of your program when you request it by issuing an ABEND macro instruction, or when the control program detects a condition that will degrade the system or destroy data.

Virtual Storage Management: While virtual storage allows you to write large programs without the need for complex overlay structures, virtual storage must be obtained for your job step. Virtual storage is allocated by both explicit and implicit requests.

Real Storage Management: The supervisor administers the use of real storage and directs the movement of virtual pages between auxiliary storage and real storage in page size blocks. The services provided allow you to release virtual storage contents, load virtual storage areas into real storage, and page out virtual storage areas from real storage.

In addition to the services outlined above, the supervisor provides the facilities for timing events, extended precision floating-point simulation, and operator communication with both the system and application programs.

Subtask Creation and Control

One task is created by the control program as a result of initiating execution of the job step (the job step task). You can create additional tasks in your program. If you do not, however, the job step task is the only task in a job being executed. The benefits of a multiprogramming environment are still available even with only one task in the job step; work is still being performed for other jobs when your task is waiting for an event, such as an input operation, to occur.

The advantage in creating additional tasks within the job step is that more tasks are competing for control than the task in the job you are concerned with. When a wait condition occurs in one of your tasks, it is not necessarily a task from some other job that gets control; it may be one of your tasks, a portion of your job.

The general rule is that parallel execution of a job step (that is, more than one task in a job step) should be chosen only when a significant amount of overlap between two or more tasks can be achieved. The amount of time taken by the control program in establishing and controlling additional tasks, and your increased effort to coordinate the tasks and provide for communications between them must be taken into account.

Most of the information concerning subtask creation and control appears in **OS/VS2 Supervisor Services and Macro Instructions**. This chapter continues discussion in the following areas:

- task creation (ATTACH macro instruction)
- communication with a problem program (EXTRACT and QEDIT macro instructions)

Creating a New Task

The ATTACH macro instruction causes the control program to create a new task. The complete use of the macro instruction is described in **OS/VS2 Supervisor Services and Macro Instructions.**

The macro instruction has nine parameters which permit the authorized user (protection key 0-7 or supervisor state) greater flexibility in using the services of the macro instruction. If the parameters are not used by authorized tasks, the default values are assigned. These defaults include:

- JSTCB=NO -- the attached task is a task in the present job step.
- SM=PROB -- the new task is to run in problem program mode.
- SVAREA=YES -- a save area is needed for the new task.
- KEY=PROP -- the protection key of the newly created task is the same as the task using ATTACH.
- DISP=YES -- the subtask is to be dispatchable.
- TID=0 -- the task identifier of the new task is 0.
- JSCB -- omission of this parameter specifies that the address of the job step control block of the attaching task is also used for the new task.
- NSHSPV and NSHSPL -- omission of these parameters specifies that subpools 236 and 237, if they exist, are to be shared with the subtask.

User Modifications

Rather than accepting the default values, (assuming the task is authorized), you can extend the facilities of the ATTACH macro instruction by coding the following values:

- JSTCB=YES -- the attached task is a new job step task. In this case, the address of the TCB of the newly created task is placed in the TCBJSTCB field of the attaching TCB.

  The first load of a job is attached as a job step task by the initiator. For such an attach, the program manager will not search the job library of the attaching task. When the job step task issues ATTACH, LOAD, LINK, or XCTL, the job library of the job step task may be searched for the load module being fetched.

  Also, only under a job step task can a system program (system key or supervisor state) attach a load module from a nonsystem library.

  In order to attach a job step task, the attaching task (and any of its subtasks) must be job step tasks. If one of these conditions is not met, the new task will not be created.

- SM=SUPV -- the system is to run in supervisor mode when executing the attached task.

  Supervisor state is a requirement before privileged instructions (for example, LPSW) can be executed. You can specify supervisor mode via this parameter or via the MODESET macro instruction.

- SVAREA=NO -- a save area is not needed for the new task.

  The save area is obtained from the user's region. Since it may not always be desirable to have a save area (for example, the user's region may not be defined at the time of a system ATTACH), this parameter may be used to specify that no save area should be created.

- KEY=ZERO -- the protection key of the newly created task is zero.

  Protection key zero allows the new task to reference any defined storage and pass all validity checks.

- DISP=NO -- the subtask is to be nondispatchable.

  This parameter causes the primary nondispatchability bit TCBANDSP to be turned on in the new TCB. As a result, the new TCB will not be dispatched. Thus, this allows the originating task to alter the new TCB. The new task will remain nondispatchable until TCBANDSP has been reset via the STATUS macro instruction with the RESET option. (*Note:* STATUS START TCB will not make the new TCB dispatchable.)

- TID=task id -- the task identifier specified is to be placed in the TCBTID field of the attached task.

  The task identifier can be set to identify critical system tasks. Other uses of this parameter are not recommended.

- JSCB=jscb address -- the address specified for the JSCB is to be used for the new task.

  This parameter sets the TCBJSCB to the address of a job step control block. This action, normally associated with the creation of a job step task, is not required by ATTACH.

- NSHSPV=subpool number and NSHSPL=subpool list address -- subpools 236 and 237 are not to be shared with the new task.

  Subpools 236 and 237 are known as the scheduler work area (SWA). This parameter allows the scheduler to control these subpools.

Operator Communication with a Problem Program

The operator can pass information to a problem program by issuing a STOP or a MODIFY command. In order to accept these commands, the program must be set up in the following manner.

An EXTRACT macro instruction is issued to obtain a pointer to the communications ECB and a pointer to the first command input buffer (CIB) on the CIB chain for the task. The ECB is posted whenever a STOP or a MODIFY command is issued. The EXTRACT macro instruction is written as follows, and will return what is indicated in Figure 1.

```
EXTRACT answer area,FIELDS=COMM
```

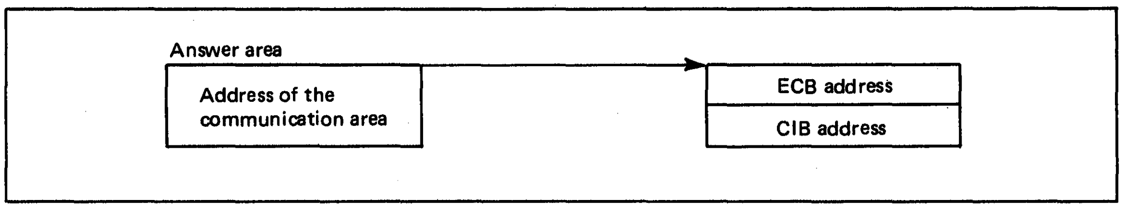


Figure 1. EXTRACT ECB and CIB Pointers

The CIB contains the information specified on the STOP or the MODIFY command, as shown in Figure 2. If the job was started from the console, the EXTRACT macro instruction will point to the START CIB. If the job was not started from the console, the address of the first CIB will be zero.

0	Address of next CIB		Verb code	CIB length	Reserved
8	Reserved	TSO terminal ID	Console ID	Reserved	Length of data field
10	Variable length data specified on the command.				

Verb code	X'04'	START
	X'40'	STOP
	X'44'	MODIFY

Figure 2. Command Input Buffer Contents

If the address of the START CIB is present, the QEDIT macro instruction should be used to free this CIB after any parameters passed in the START command have been examined. The QEDIT macro instruction is written as follows:

```
QEDIT ORIGIN=address of pointer to CIB,BLOCK=address of CIB
```

The CIB counter should then be set to allow CIBs to be chained and MODIFY commands will be accepted for the job. This is also accomplished by using the QEDIT macro instruction:

```
QEDIT ORIGIN=address of pointer to CIB,CIBCTR=n
```

The value of n is any integer value from 0 to 255. If n is set to zero, no MODIFY commands will be accepted for the job. STOP commands, however, will be accepted for the job regardless of the value set for CIBCTR.

For the duration of the job, the communications ECB may be waited on or checked at any time to see if a command has been entered for the program. The verb code in the CIB should be examined to determine whether a STOP or a MODIFY command has been entered. After the data in the CIB has been processed, a QEDIT macro instruction should be issued to free the CIB.

The communications ECB will be cleared when no more CIBs remain. Care should be taken if multiple subtasks are examining these fields. Any CIBs not freed by the task will be unchained by the system when the task is terminated. The area addressed by the pointer obtained by the EXTRACT macro instruction, the communications ECB, and all CIBs are in protected storage and may not be altered.

Providing an EXTRACT Answer Area

The EXTRACT macro instruction is used to provide TCB information for either the active task or one of its subtasks. The information from the requested field is returned in the relative order shown in Figure 3. If the information from a field is not requested, the associated fullword is omitted.

Answer Area Address

GRS	00	Address		
FRS	00	Address		
Reserved	00	00	00	00
AETX	00	Address		
PRI	00	00	Value	Value
CMC	00	Completion Code		
TIOT	00	Address		
COMM	00	Address		
TSO	00	Address		
PSB	00	Address		
TJID	00	00	Value	
ASID	00	00	Value	
	1 Byte	1 Byte	1 Byte	1 Byte

Figure 3. EXTRACT Answer Area Fields

You must provide an answer area consisting of contiguous fullwords, one for each of the codes specified in the FIELDS parameter, with the exception of ALL. If ALL is specified, you must provide a 7-word area to accomodate the GRS, FRS, reserved, AETX, PRI, CMC, and TIOT fields. The ALL code does not include the COMM, TSO, PSB, TJID, and ASID fields.

Addresses are always returned in the low-order three bytes of the fullword, and the high-order byte is set to zero. Fields for which no address or value has been specified in the task control block are set to zero.

For example, if FIELDS=(TIOT,GRS,PRI,TSO,PSB,TJID) is coded, a 6-fullword answer area is required, and the extracted information will appear in the same relative order as shown in Figure 3. (That is, GRS will be returned in the first word, PRI in the second word, TIOT in the third word, etc.)

If FIELDS=(ALL,TSO,PSB,COMM,ASID) is coded, an 11-fullword answer area is required, and the extracted information will appear in the answer area in the relative order shown above.

Program Management

The supervisor can be used to aid communication between segments of a program. The descriptions of load module structures, facilities for passing control between programs, and the use of associated macro instructions are available to all users and are described in **OS/VS2 Supervisor Services and Macro Instructions.**

Following is a description of synchronous exits and the SYNCH macro instruction associated with it. The facility should only be used by system programmers or other installation-approved personnel.

Synchronous Exits

In general, the SYNCH macro instruction is used when a control program in the supervisor state is to give temporary control to a processing program routine, and when the processing program is expected to return control to the supervisor state. The program to which control is given must be in virtual storage when the macro instruction is issued. When the processing program returns control, the supervisor state bit, the storage protection key bits, the system mask bits, and the program mask bits of the program status word are restored to the settings they had before execution of the SYNCH macro instruction.

The use of the SYNCH macro instruction is similar to that of the BALR instruction in that register 15 may be used for the entry name address.

As an example of the use of the SYNCH macro instruction, label processing as the result of an OPEN macro instruction may be carried out to a point at which a user's processing program indicates that private processing is desired (or necessary). The control program's Open routine would then issue a SYNCH macro instruction giving the entry name address of the subroutine required for the user's private label processing.

Multiprocessing Programming Considerations

Many of the following programming considerations are true in a multi-tasking environment (such as OS/MVT) as well as in a multiprocessing system. However, because of their increased importance in a multiprocessing environment, they should be closely reviewed.

Checkpoint/Restart

When issuing checkpoints and then restarting a task, the restarted task must request control of all resources required to continue processing. Resources are not automatically returned to the task upon restart.

Re-entrant Modules

When link editing modules as re-entrant, be sure that all the modules and the macro instructions they call are re-entrant. This is important since in a multiprocessing system:

- two tasks in the same address space making use of the module might cause the module to be executed simultaneously on two different CPUs.
- asynchronous appendages can operate on one CPU simultaneously with an associated task on the other CPU.
- recovery routines can execute on either CPU, not necessarily on the one on which the error was detected.

The CSECTs must be unchanged during execution or their critical sections must be explicitly serialized. The general method for ensuring re-entrancy of macros is to use the LIST and EXECUTE forms of the macro instructions with a dynamically acquired parameter list.

Priority

Programs that use priority or precedence as a serialization mechanism are sensitive to changes in the dispatching algorithms used by the system. For example, the CHAP macro instruction will not ensure that tasks are dispatched in the expected order, due to dispatching on two CPUs. Also, the PRIORITY and DPRTY JCL parameters can no longer be used to accomplish serialization. First, the system resources manager might allow a task or job with a lower dispatching priority to execute prior to a task with a higher priority. Second, since tasks are executed on both CPUs, tasks of different priority might be executed on both CPUs, tasks of different priority might be executed on both CPUs simultaneously.

Note: Tasks might not be redispatched on the CPU on which they were previously executing. Therefore, storage from 0-4K should not be used because redispatch on a different CPU will result in different data being referenced.

For further multiprocessing considerations see the section "Locking".

Resource Control

Planning is required when more than one program uses a serially reusable resource. Planning is also required when portions of some tasks depend on the completion of events in other tasks.

This chapter discusses some of the services available to control resources, and thus to help you plan ahead for a more efficient installation. The services discussed include:

- locking (SETLOCK macro instruction)
- must complete function (ENQ and DEQ macro instructions)
- shared DASD (RESERVE and EXTRACT macro instructions)
- authorized program facility (TESTAUTH macro instruction)
- system status (MODESET macro instruction)
- interprocessor communications (DSGNL, RISGNL, and RPSGNL macro instructions)
- event completion (POST, SPOST, and EVENTS macro instructions)

Locking

Adequate control of serially reusable resources is a significant problem in a multiprocessing environment. Since some uniprocessor serialization techniques (for example, disablement), are no longer effective, there is the possiblity of multiple tasks, even multiple tasks in the same job step, using the same serially reusable resource while running concurrently on different CPUs.

To eliminate this problem, a locking mechanism is provided to control access to serially reusable resources. The lock manager will handle all functions related to the locks (for example, obtaining or releasing locks, or checking the status of a CPU with respect to a particular lock). However, a locking technique can not be effective unless all routines that have the potential for changing the protected resource, or that depend on its status remaining unchanged for a given period, make use of the locking mechanism.

In MVS, a locking manager controls a new hierarchical locking structure with multiple types of locks and synchronizes the use of serially reusable resources. Use of the locking manager is restricted to key 0 programs running in supervisor state, which prevents unauthorized problem programs from interfering with the serialization process.

To enhance performance, each kind of serially resuable resource is assigned a separate lock. In this way, a lock held by a CPU on one resource does not prevent the other CPU from using a different resource.

The locks provided in MVS are:

- Global dispatcher lock (DISP) -- serializes all functions associated with the dispatching process.
- Auxiliary storage manager lock (ASM) -- serializes use of the global ASM control blocks.
- Real storage manager and virtual storage manager space allocation lock (SALLOC) -- serializes the global functions of real storage management and virtual storage management.
- IOS synchronization lock (IOSYNCH) -- serializes global IOS functions.
- IOS channel availability table lock (IOSCAT) -- serializes access and updates to the channel availability table.
- IOS unit control block lock (IOSUCB) -- serializes access and updates to the unit control blocks. There is one lock per UCB.
- IOS logical channel queue lock (IOSLCH) -- serializes access and updates to the IOS logical channel queues. There is one lock per channel queue.

- System resources manager lock (SRM) -- serializes use of the SRM control algorithms and associated data.
- Cross memory services lock (CMS) -- serializes on more than one address space where this serialization is not provided by one or more of the other global locks.
- Local storage lock (LOCAL) -- serializes functions and storage, used by the local supervisor, within local address space. There is one lock per address space.

Classes of Locks

Two classes of locks exist:

- Global locks -- protect serially resuable resources related to more than one address space. (For example, a unit control block is protected by a global lock since it relates to the entire system. Also, a system-related GETMAIN for a global subpool, or a global ENQ, requires a global lock.)
- Local locks -- protect the resources assigned to a particular address space. When a CPU holds a local lock, the queues and control blocks associated with that address space can be manipulated only by the CPU holding the lock. (For example, an address space-related GETMAIN for a user subpool requires a local lock.)

All of the locks described above, with the exception of the local lock, are global locks. These global locks provide system-wide services or use control information in the common area and must serialize across address spaces. The local locks, on the other hand, do not serialize across address spaces, but serialize functions executing within the address space.

Types of Locks

Two types of locks exist. The type determines what happens when a CPU makes an unconditional request for a lock that is held by the other CPU. The types are:

- Spin locks -- prevent the requesting CPU from doing any work until the lock is cleared by the other CPU. The requesting CPU enters a loop that keeps testing the lock until the other CPU releases it. As soon as the resource is free, the first CPU can obtain the resource and continue processing.
- Suspend locks -- prevent the requesting program from doing work until the lock is available, but allow the CPU to continue doing other work. The request is queued and the requesting CPU is dispatched to do other work. Upon release of the lock, the highest priority queued requestor will be given control of the lock.

All of the locks described above, with the exception of the local and cross memory services locks, are spin locks. The LOCAL and CMS locks run enabled and can be interrupted to run higher priority work. If there is another request for the lock while it is held, the requestor is suspended and other work is dispatched.

The CMS lock was provided as an enabled global lock for the following reasons:

- Since disabled page faults are not allowed in the system, some global functions could use a lock which did not require the functions to fix all their code and control blocks.
- Some functions require significant amounts of time under the lock and could impact the responsiveness of the system. By running these functions logically disabled under the lock, responsiveness is retained at the expense of some increased contention for the lock.

The other locks were left as disabled spin locks because normally the functions which run under the locks are of short duration, and the cost in system overhead to perform the status saving necessary to accept interruptions and allow switching would offset the gain in responsiveness. Also, the more frequently used functions (for example, IOS interruption handler, dispatcher, and storage manager) are needed to perform interruption handling and task switching, and thus have to remain disabled.

If a lock is unconditionally requested, the lock will be unconditionally obtained. If the lock is conditionally requested, the requestor will be given the lock if it is available; if the lock is unavailable, control will be returned to the caller without the lock. (See the COND and UNCOND parameters on the SETLOCK macro instruction.)

Figure 4 summarizes the characteristics of the locks.

lock	global	local	spin	suspend
DISP	X		X	
ASM	X		X	
SALLOC	X		X	
IOSYNCH	X		X	
IOSCAT	X		X	
IOSUCB	X		X	
IOSLCH	X		X	
SRM	X		X	
CMS	X			X
LOCAL		X		X

Figure 4. Summary of Locking Characteristics

Locking Hierarchy

To prevent a deadlock between CPUs, the locks are arranged in a hierarchy, and a CPU may unconditionally request only locks higher in the hierarchy than locks that it currently holds. The locking hierarchy is the order in which the locks are listed in Figure 4, with DISP being the highest lock in the hierarchy.

As noted above, some locks are single system locks (for example, DISP), and some locks are multiple locks in which there is more than one lock within the lock type (for example, IOSUCB).

For those global lock types which have more than one lock, a CPU may only hold one lock of each type. For example, if a CPU holds an IOSUCB lock, it may not request a different IOSUCB lock.

The LOCAL lock must be held by the caller when requesting the CMS lock. Also, the LOCAL lock cannot be released while holding the CMS lock.

It is not necessary to obtain all locks in the hierarchy up to the highest lock needed. Although only the needed locks have to be obtained, they do, however, have to be obtained in hierarchy sequence.

Locking Considerations

In MVS the locking function is provided to replace disabling in the control program with a form of logical disabling which works across both CPUs. Locking, however, is not completely equivalent to disablement in VS2 Release 1, and required some changes in the system. The significant differences are:

- In VS2 Release 1, a disabled program could issue an SVC and receive control back disabled. In MVS, a locked routine is not allowed to issue an SVC.
- All user-written functions which disable using the MODESET macro instruction, the SSM (set system mask) instruction, or some other method, should be changed to use the locking function. The SYSMASK and ENABLE parameters of MODESET are no longer supported; the SSM instruction causes a program check.

- In VS2 Release 1, disablement prevented storage from being freed or paged out while the storage was being referenced. In MVS, it is necessary to hold the local lock to prevent a FREEMAIN from being executed on the other CPU even thought a global spin lock is held on one CPU. To prevent page stealing, it is necessary to either fix the pages or hold the SALLOC lock.

Resource Serialization

Resource serialization is a potential source of integrity problems. An integrity exposure can result when serialization controls on sensitive control program resources are nonexistent, or inadequate. That is, an unauthorized problem program can directly, or indirectly, through a part of the control program, change the content or status of a system resource while another portion of the control program is using that resource or is in some way dependent on its content or status remaining unchanged for a given period of time. An example of this exposure would be if an unauthorized problem program issued an SVC and changed the content of a control block while another SVC was using that same control block (resource).

Adequate control of serial resources becomes even more significant in a multiprocessing environment since some uniprocessor serialization techniques (such as disablement) are no longer effective because of the possibility of multiple tasks using the same serial resource and running concurrently on different CPUs.

Serialization Considerations

To eliminate the potential exposures described above, a locking technique is used to serialize access to the resources in question. This locking technique is only effective, however, if all routines that have the potential for chaiging the resource, or that depend on its status remaining unchanged for a given period, make use of the same locking mechanism. Effective use of a locking technique, therefore, requires considerable investigation and effort. You must both determine and then keep track of system resources that must be serialized and the routines that access such resources.

Another integrity problem is the time-of-check-to-time-of-use consideration. That is, from the time of a validity check, until the completion of the operation associated with that validity check, the variables on which the outcome of the validity check is based must not be allowed to change to the extent that the result of the validity check would be changed. While this is a relatively obvious statement, the requirement it imposes has considerably influenced the direction of the integrity support in MVS.

In some cases, this requirement can be met automatically through the use of hardware serialization mechanisms such as checking the validity of fetch/store operations by assuming the user's protenction key. In other cases, techniques such as saving validated data in protected storage (storage keys 0-7) and use of the previously described locking mechanism are required.

Serialization Requirements

The only safe serialization is explicit serialization by means of one of the following methods: ENQ/DEQ; WAIT/POST/EVENTS; RB precedence at the TCB level; locking at the TCB level; CS (compare and swap instruction); and TS (test and set instruction). Such forms of serialization are required in the following cases:

- Scanning of the CIB chain. Manipulation of the CIB chain should be done by means of the QEDIT macro instruction.
- The use of data in subpools shared between tasks.
- Data referenced by more than one task. (For example, attached tasks can execute at the same time as the mother task on different CPUs.)

- References to system control block fields that dynamically change after IPL. The serialization technique in this case must match that used by the system. Also, bits within a byte all require the same serialization technique, if the byte might be updated simultaneously by both CPUs.
- The access of data sets shared between tasks in the same address space, if the tasks update the data and if the access method is not VSAM or BDAM.
- Any common data references between an ESTAE exit and asynchronous exits, if ESTAE with ASYNCH=YES is issued.

SETLOCK Macro Instruction

The SETLOCK macro instruction is used to obtain, release, or test a specified lock or set of locks (using the OBTAIN, RELEASE, and TEST parameters). To use SETLOCK, you must be executing in supervisor state with protection key 0.

Disabled/Enabled State for OBTAIN: When a global spin lock is successfully obtained, control is returned to the SETLOCK caller with the CPU in a physically disabled state (except for machine check interruptions).

When a LOCAL lock or CMS lock is successfully obtained, control is returned to the SETLOCK caller with the CPU in the physically enabled or disabled state that existed prior to the SETLOCK request. This is also true for unsuccessful attempts to obtain any of the locks where control is returned to the caller.

Disabled/Enabled State for RELEASE: When a global spin lock is successfully released, control is returned to the caller with the CPU in a physically enabled state if no other global spin locks are held by that CPU; control is returned in a physically disabled state if other global spin locks are held. Control will also be returned in the disabled state if any disabled supervisor indicators are on when the RELEASE request is made or if the DISABLED parameter was specified.

When the LOCAL lock or CMS lock is released, control is returned with the CPU in the enabled or disabled state that existed prior to the RELEASE request.

For all locks, if the RELEASE operation is unsuccessful and no ABEND condition exists, control is returned with the CPU in the physically enabled or disabled state that existed prior to the RELEASE request.

For multiple RELEASE requests (via SPIN, ALL, or (reg) subparameters), the final state is the same as what would have existed had the locks been released one at a time.

The Must Complete Function

System routines (routines operating under a storage protection key of zero) often engage in updating and/or manipulation of system resources such as system data sets, control blocks, and queues. These resources contain information critical to continued operation of the system. The system routines must complete their operations on the resource; otherwise, the resource may be left incomplete or may contain erroneous information.

The ENQ service routine ensures that a routine queued on a critical resource(s) can complete processing of the resource(s) without interruptions leading to termination. ENQ can place other tasks in a wait state until the requesting task -- the task issuing a ENQ macro instruction with the set-must-complete (SMC) parameter -- has completed its operations on the resource. The requesting task releases the resource and terminates the must complete condition by issuing a DEQ macro instruction with the reset-must-complete (RMC) parameter.

Because the must complete function serializes operations to some extent, its use should be minimized -- use the function only in a routine that processes system data whose validity must be ensured.

As an example, in multitask environments the integrity of the volume table of contents (VTOC) must be preserved during updating so that all future users may have access to the latest, correct version of the VTOC. Thus, in this case, ENQ on the VTOC and use the must complete function (to suspend processing of other tasks) when updating a VTOC.

Just as the ENQ function serializes use of a resource requested by many different tasks, the must complete function serializes execution of tasks.

Characteristics of the Must Complete Function

The must complete function can be used only at the step level, where only the current task in an address space is allowed to execute. All other tasks, including the initiator task, are placed in a wait state.

When the must complete function is requested, the requesting task is marked as being in the must complete mode when the resource(s) queued upon are available. All asynchronous exits from the requesting task are deferred. The initiator and all other tasks in the job step are set nondispatchable. Thus, tasks external to the requesting task are prevented from initiating procedures that will cause termination of the requesting task. Other external events, such as a CANCEL command issued by an operator, or a job step time expiration, are also prevented from terminating the requesting task.

The failure of a task which owns a must-complete resource results in the abnormal termination of the entire job step. The programmer receives a message stating that the failure occurred in the step must complete status.

Programming Notes

1. All data used by a routine that is to operate in the must complete mode should be checked for validity to ensure against a program-check interruption.
2. If a routine that is already in the must complete mode calls another routine, the called routine also operates in the must complete mode. An internal count is maintained of the number of SMC requests; an equivalent number of RMC requests is required to reset the must complete function.
3. Interlock conditions can arise with the use of the ENQ function. Additionally, an interlock may occur if a routine issues an ENQ macro instruction while in the must complete mode. The wanted resource may already be queued on by a task made nondispatchable due to the must complete request already made. Since the resource cannot be released, job step tasks are abnormally terminated, and system tasks are retried.
4. The macro instructions ATTACH, LINK, LOAD, and XCTL should not be used, *unless extreme care is taken,* by a routine operating in the must complete mode. An interlock condition will result if a serially reusable routine requested by one of these macro instructions has been requested by one of the tasks made nondispatchable by the use of the SMC parameter or was requested by another task and has been only partially fetched.
5. The time a routine is in the must complete mode should be kept as short as possible -- enter at the last moment and leave as soon as possible. One suggested way is to:

   a. ENQ (on desired resource(s))

b.ENQ (on same resource(s)),RET=HAVE,SMC=STEP

Item a gets the resource(s) without putting the routine into the must complete mode. Later, when appropriate, issue the ENQ with the must complete request (Item b). Issue a DEQ macro instruction to terminate the must complete mode as soon as processing is finished.

6. The STATUS macro instruction changes the dispatchability status of tasks for users in supervisor state, with a protection key of 0-7, or APF-authorized. STATUS can also change the must complete status of a task. In all cases, the task remains enqueued.

Tasks set nondispatchable by the corresponding ENQ macro instruction are made dispatchable and asynchronous exits from the requesting task are enabled.

Shared Direct Access Storage Devices (Shared DASD)

The Shared DASD facility allows computing systems to share direct access storage devices. Systems can share common data and consolidate data when necessary. No change to existing records, data sets, or volumes is necessary to use the facility. However, reorganization of volumes may be desirable to achieve better performance.

Devices that Can be Shared

The following control units and devices are supported by the Shared DASD option:

- IBM 2314 Direct Access Storage Facility equipped with the two-channel switch -- IBM 2314 Disk Storage Module.
- IBM 2314 Direct Access Storage Facility combined with the IBM 2844 Auxilliary Storage Control -- IBM Disk Storage Module. Device reservation and release are supported by this combination with or without the presence of the two-channel switch. Two channels -one from System A and one from System B -- may be connected to the combination. In addition, the two-channel switch may be installed in either or both of the control units, thus permitting as many as four systems to share the devices.
- IBM 2835 Storage Control Unit with two-channel switch -IBM 2305 Fixed Head Storage Facility.
- IBM 3830 Storage Control Unit with two-channel switch -- IBM 3330 Series Disk Storage Drive. The IBM 3330, 3340/3344, 3350 Series devices may also be configured for shared use via the string switch feature.

Alternate channels to a device from any one system may only be specified for the IBM 2314 Direct Access Storage Facility, or the IBM 3330, 3340/3344, 3350 Series Storage Unit.

Volume/Device Status

The Shared DASD facility requires that certain combinations of volume characteristics and device status be in effect for shared volumes of devices. Figure 5 shows the combinations that must be in effect for a volume or device:

System A	Systems B, C, D
Permanently resident	Permanently resident
Reserved	Reserved
Removable	Offline
Offline	Removable, reserved, or permenently resident

Figure 5. Valid Volume Characteristic and Device Status Combinations

If a volume/device is marked removable on any one system, the device must be in offline status on all other systems. The mount characteristic of a volume and/or device status may be changed on one system as long as the resulting combination is valid for other systems sharing the device. No other combination of volume characteristics and device status is supported.

System Configuration

Operating system configurations do not have to be identical to share a data set. The only additional equipment needed for the Shared DASD option is either a two-channel switch or a 2844 Auxiliary Control unit. The user must also observe certain restrictions about the data sets that are shared. The following data sets cannot be shared:

```
PASSWORD         SYS1.NUCLEUS
SYS1.LOGREC      SYS1.SVCLIB
SYS1.LPALIB      VSAM page spaces
SYS1.MANX        Master catalog
SYS1.MANY
```

Volume Handling

Volume handling on the Shared DASD option must be clearly defined since operator actions on the sharing system must be performed in parallel. The following rules should be in effect when using the Shared DASD option:

- Operators should initiate all shared volume mounting and demounting operations. The system will dynamically allocate devices unless they are in reserved or permanently resident status, and only the former can be changed by the operator.
- Mounting and demounting operations must be done in parallel on all sharing systems. A VARY OFFLINE must be effected on all systems before a device may be dismounted.
- Valid combinations of volume mount characteristics and device status for all sharing systems must be maintained. To IPL a system, a valid combination must be established before device allocation can proceed. This valid combination is established either by specifying mount characteristics of shared devices in VATLST, or varying all sharable devices offline prior to issuing START commands and then following parallel mount procedures.

Macro Instructions Used with Shared DASD

The RESERVE macro instruction is used to reserve a device for use by a particular system; it must be issued by each task needing device reservation. The EXTRACT macro instruction is used to obtain the address of the task input/output table (TIOT) from which the UCB address can be obtained. The topic "Finding the UCB Address" explains this procedure.

Notes on RESERVE: The Set-Must-Complete (SMC) parameter available with the ENQ macro instruction may also be used with RESERVE.

If a restart occurs when a RESERVE is in effect for devices, the system will not restore the RESERVE; the user's program must reissue the RESERVE.

Releasing Devices

The DEQ macro instruction is used in conjunction with RESERVE just as it is used with ENQ. It must describe the same resource and its scope must be stated as SYSTEMS; however, the UCB=pointer address parameter is not required. If the DEQ macro instruction is not issued by a task which has previously reserved a device, the system will free the device when the task is terminated.

Preventing Interlocks

The more often device reservations occur in each sharing system, the greater the chance of interlocks occurring. Allowing each task to reserve only one device minimizes the exposure to interlock. The system cannot detect interlocks caused by program use of the RESERVE macro instruction and enabled wait states will occur on the system or systems.

Volume Assignment

Since exclusive control is by device, not by data set, consider which data sets reside on the same volume. In this environment it is quite possible for two tasks in two different systems -- processing four different data sets on two shared volumes -- to become interlocked. For example, as shown in Figure 6, data sets A and B reside on device C, and data sets D and E reside on device F. Task X in system X reserves device C in order to use data set A; task Y in system Y tries to reserve device F in order to use data set D. Now task X in system X tries to reserve device F in order to use data set E and task Y in system Y tries to reserve device C in order to use data set B. Neither can ever regain control, and neither will complete normally. When the system has job step time limits, the task, or tasks, in the interlock would be abnormally terminated when the time limit expires. Moreover, an interlock could mushroom, encompassing new tasks as these tasks try to reserve the devices involved in the existing interlock.

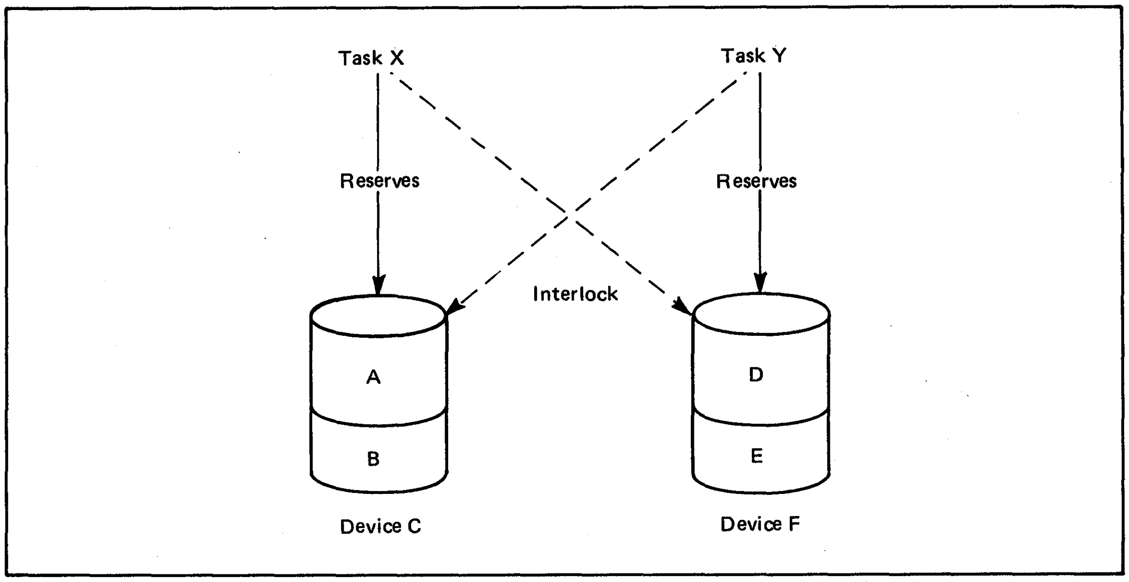


Figure 6. Example of an Interlock Environment

Program Libraries

When assigning program libraries to shared volumes, precaution must be taken to avoid interlock. For example, LINKLIB for system A resides on volume X, while LINKLIB for system B resides on volume Y. Task A in system A invokes a direct access device space management function for volume Y, resulting in that device being reserved. Task B in system B invokes a similar function for volume X, reserving that device. However, each load module transfers to another load module via XCTL. Since the LINKLIB for each system resides on a volume reserved by the other system, the XCTL macro instruction cannot complete the operation. An interlock occurs; since no access to LINKLIB is possible, both systems will eventually enter an enabled wait state.

Finding the UCB Address

This topic explains procedures for finding the UCB address for use by the RESERVE macro instruction; it also shows a sample assembler language subroutine which issues the RESERVE and DEQ macro instructions and can be called by higher level languages.

Providing the Unit Control Block Address to RESERVE: The EXTRACT macro instruction is used to obtain information from the task control block (TCB). The address of the TIOT can be obtained from the TCB in response to an EXTRACT. Prior to issuing an EXTRACT macro instruction, the user sets up an answer area in main storage which is to receive the requested information. One full word is required for each item to be provided by the control program. If the user wishes to obtain the TIOT address, he must specify FIELDS=TIOT in the EXTRACT macro instruction.

The address of the TIOT is then returned by the control program, right adjusted, in the full word answer area.

The TIOT is constructed by job management routines and resides in virtual storage during step execution. The TIOT consists of one or more DD entries, each of which represents a data set defined by a DD statement for the jobstep. Each entry includes the DD name. Associated with each DD entry is the UCB address of the associated device. In order to find the UCB address, the user must locate the DD entry in the TIOT corresponding to the DD name of the data set for which he intends to issue the RESERVE macro instruction.

The UCB address can also be obtained via the data extent block (DEB) and the data control block (DCB). The DCB is the block within which data pertinent to the current use of the data set is stored. The address of the DEB is contained at offset 44 decimal after the DCB has been opened. The DEB contains an extension of the information in the DCB. Each DEB is associated with a DCB and the two point to each other.

The DEB contains information concerning the physical characteristics of the data set and other information that is used by the control program. A device dependent section for each extent is included as part of the DEB. Each such extent entry contains the UCB address of the device to which that portion of the data set has been allocated. In order to find the UCB address, the user must locate the extent entry in the DEB for which he intends to issue the RESERVE macro instruction. (In disk addresses of the form MBBCCHHR, the M indicates the extent number starting with 0).

Procedures for Finding the UCB Address of a Reserved Device: For data sets using the queued access methods in the update mode or for unopened data sets:

1. Extract the TIOT from the TCB.
2. Search the TIOT for the DD name associated with the shared data set.
3. Add 16 to the address of the DD entry found in step 2. This results in a pointer to the UCB address in the TIOT.
4. Issue the RESERVE macro specifying the address obtained in step 3 as the parameter of the UCB keyword.

For opened data sets:

1. Load the DEB address from the DCB field labeled DCBDEBAD.
2. Load the address of the field labeled DEBDVMOD in the DEB obtained in step 1. The result is a pointer to the UCB address in the DEB.
3. Issue the RESERVE macro specifying the address obtained in step 2 as the parameter of the UCB keyword.

For BDAM data sets the user may reserve the device at any point in the processing in the following manner:

1. Open the data set.
2. Convert the block address used in the READ/WRITE macro to an actual device address of the form MBBCCHHR.
3. Load the DEB address from the DCB field labeled DCBDEBAD.
4. Load the address of the field labeled DEBDVMOD in the DEB.
5. Multiply the "M" of the direct access by 16.
6. The sum of steps 4 and 5 is the address of the correct extent entry in the DEB for the next READ/WRITE operation. The sum is also a pointer to the UCB address for this extent.
7. Issue the RESERVE macro specifying the address obtained in step 6 as the parameter of the UCB keyword.

If the data set is an ISAM data set, QISAM in the load mode should be used only at system update time. Further, if it is a multivolume ISAM data set, it must be assumed that all jobs will access the data set through the highest level index. The indexes should never reside in virtual storage when the data set is being shared. In this case, by issuing a RESERVE macro for the volume on which the highest level index resides, the user effectively reserves the volumes on which the prime data and independent overflow areas reside. The following procedures can be used to achieve this:

1. Open the data set.
2. Locate the actual device address (MBBCCHHR) of the highest level index. This address can be obtained from the DCB.
3. Load the DEB address from the DCB field labeled DCBDEBAD.
4. Load the address of the field labeled DEBDVMOD in the DEB.
5. Multiply the "M" of the actual device address located in step 2 by 16.
6. The sum of steps 4 and 5 is the address of the correct extent entry in the DEB for the next READ/WRITE operation. The sum is also a pointer to the UCB address for this extent.
7. Issue the RESERVE macro specifying the address obtained in step 6 as the parameter of the UCB keyword.

RES and DEQ Subroutines: The assembler language subroutine in Figure 7 can be used by assembler language programs to issue the RESERVE and DEQ macro instructions. Parameters that must be passed to the RESDEQ routine, if the RESERVE macro instruction is to be issued, are:

DDNAME - the eight character name of the DDCARD for the device to be reserved.

QNAME - an eight character name.

RNAME LENGTH - one byte (a binary integer) that contains the RNAME length value.

RNAME - a name from 1 to 255 characters in length.

The DEQ macro instruction does not require the UCB=pointer address as a parameter. If the DEQ macro is to be issued, a fullword of binary zeros must be placed in the DDNAME field before control is passed.

```
RESDEQ      CSECT
            SAVE    (14,12),T           SAVE REGISTERS
            BALR    2,0                 SET UP ADDRESSABILITY
            USING   *,2
            ST      13,SAVE+4
            LA      11,SAVE             ADDRESS OF MY SAVE AREA IS
                                        STORED IN THIRD WORD OF CALLER'S
            ST      11,8(13)            SAVE AREA
            LR      13,11               ADDRESS OF MY SAVE AREA
            LR      9,1                 ADDRESS OF PARAMETER LIST
            L       3,0(9)              DDNAME PARAMETER OR WORD OF
                                        ZEROS
            CLC     0(4,3),=F'0'        WORD OF ZEROS IF DEQ IS
                                        REQUESTED
            BE      WANTDEQ
*PROCESS FOR DETERMINING THE UCB ADDRESS USING THE TIOT
            XR      11,11               REGISTER USED FOR DD ENTRY
            EXTRACT ADDRTIOT,FIELDS=TIOT
            L       7,ADDRTIOT          ADDRESS OF TASK I/O TABLE
            LA      7,24(7)             ADDRESS OF FIRST DD ENTRY
NEXTDD      CLC     0(8,3),4(7)         COMPARE DDNAMES
            BE      FINDUCB
            IC      11,0(7)             LENGTH OF DD ENTRY
            LA      7,0(7,11)           ADDRESS OF NEXT DD ENTRY
            CLC     0,(4,7),=F'0'       CHECK FOR END OF TIOT
            BNE     NEXTDD
            ABEND   200,DUMP            DDNAME IS NOT IN TIOT, ERROR
FINDUCB     LA      8,16(7)             ADDRESS OF WORD IN TIOT THAT
*                                       CONTAINS ADDRESS OF UCB
*PROCESS FOR DETERMINING THE QNAME REQUESTED
WANTDEQ     L       7,4(9)              ADDRESS OF QNAME LENGTH
            MVC     QNAME(8),0(7)       MOVE IN QNAME
*PROCESS FOR DETERMINING THE RNAME AND THE LENGTH OF RNAME
            L       7,8(9)              ADDRESS OF RNAME LENGTH
            MVC     RNLEN+3(1),0(7)     MOVE BYTE CONTAINING LENGTH
            L       7,RNLEN
            STC     7,RNAME             STORE LENGTH OF RNAME IN THE
*                                       FIRST BYTE OF RNAME PARAMETER
*                                       FOR RES/DEQ MACROS
            L       6,12(9)             ADDRESS OF RNAME REQUESTED
            BCTR    7,0                 SUBTRACT ONE FROM RNAME LENGTH
            EX      7,MOVERNAM          MOVE IN RNAME
            CLC     0(4,3),=F'0'
            BE      ISSUEDEQ
            RESERVE (QNAME,RNAME,E,0,SYSTEMS),UCB=(8)
            B       RETURN
ISSUEDEQ    DEQ     (QNAME,RNAME,0,SYSTEMS)
RETURN      L       13,SAVE+4           RESTORE REGISTERS AND RETURN
            RETURN  (14,12),T
            BCR     15,14
MOVERNAM    MVC     RNAME+1(0),0(6)
ADDRTIOT    DC      F'0'
SAVE        DS      18F
QNAME       DS      2F
RNAME       DS      CL256
RNLEN       DC      F'0'
            END
```

Figure 7. Example of Subroutine Issuing RESERVE and DEQ

Authorized Program Facility (APF)

The authorized program facility (APF) provides two services. First, it limits the use of sensitive system SVC service routines and, optionally, sensitive user SVC service routines to authorized programs; that is, to system or user problem programs that are under the control of the installation manager. If an installation has its own special SVCs that bypass normal system security and integrity checks, then the APF facility should be used to restrict their use to authorized programs. Second, it ensures that load modules accessed by authorized programs, via the LINK, XCTL, LOAD, and ATTACH macro instructions, are always from authorized libraries.

A program is authorized if it is executing in supervisor state, is executing in a system key (0-7), or is executing as a part of an APF-authorized job step. A job step is APF-authorized if the first program executed in the step is from an APF-authorized library and it is link edited with the APF-authorization code (AC=1).

Note: If a job step is established APF-authorized, then it will stay APF-authorized until it completes. Similarly, if a job step is established non-APF-authorized, it will stay non-APF-authorized until it completes.

Restricting SVC Service Routines

MVS provides two methods of restricting sensitive SVC service routines:

- Sensitive SVCs may be restricted to authorized job steps via the FC01 parameter on the SVCTABLE macro instruction issued at system generation time. The SVC first level interrupt handler will ensure that SVC routines so restricted are accessible only to authorized programs. A 047 abend will result if an unauthorized program attempts to access a restricted SVC service routine. Sensitive IBM-supplied SVCs are automatically restricted.
  Note: The user of APF should be aware that APF authorization is defined such that any SVC that is restricted by the APF mechanism can be executed by any system key (0-7) or supervisor state routine. However, this is not true in reverse. A program executing as part of an APF-authorized step will not automatically be allowed system services that are restricted by system key or privileged mode tests.
- When only a portion of an SVC's function is sensitive, then the TESTAUTH macro instruction can be used to restrict particular paths through the SVC. TESTAUTH tests for the presence of system key (0-7), supervisor state, and APF authorization; if any of these conditions is present, the SVC invoker is authorized. The TESTAUTH macro cannot be used to control the use of I/O appendages. Appendages are controlled by means of the IEAAPP00 member of SYS1.PARMLIB. (See the description of this member in the **Initialization and Tuning Guide.**) The TESTAUTH macro instruction, inserted at appropriate locations in the SVC, returns an authorizated or unanthorized indication. The SVC should take appropriate action based upon this return.

Restricting Load Module Access

APF automatically prohibits access by authorized programs via the LINK, LOAD, XCTL, and ATTACH macro instructions to any load module which is not form an APF-authorized library. A 306 abend will result if the only available copy of a module accessed by an authorized program resides on a non-APF-authorized library. It is the responsibility of the authorized program to recover from the 306 abend in a way which would not allow the accessed module to execute.

As long as a load module resides on an APF-authorized library, it may be accessed by an authorized program. To preclude access to an incorrect module (e.g., via a STEPLIB), it is the installation's responsibility to ensure that duplicate module names are not permitted across APF-authorized libraries.

Defining APF-Authorized Libraries

A module must reside on an APF-authorized library if it is to be accessed, via LINK, XCTL, LOAD, or ATTACH, by an authorized program. SYS1.LINKLIB and SYS1.SVCLIB are automatically considered as APF-authorized. SYS1.LPALIB and SYS2.LPALIB are also automatically considered APF-authorized, but only during system initialization when the link pack areas are built.

Any library may be designated as an APF-authorized library by inclusion of its name in the SYS1.PARMLIB member IEAAPF00 or IEAAPFxx prior to IPL. It is the installation's responsibility to control the contents of such libraries.

If libraries are concatenated in a JOBLIB or STEPLIB, then the concatenation will be considered APF-authorized only if each and every library in the concatenation is APF-authorized.

Any library that is concatenated to SYS1.LINKLIB and contained in LNKLSTxx is designated APF-authorized. However, if the library is specified separately via JOBLIB or STEPLIB, it is no longer considered authorized unless it is also included in IEAAPFxx.

Link Edit APF-Authorization Code

For a job step to execute APF-authorized, the first program executed in the step must not only be from an APF-authorized library, but it must also be link edited with the APF-authorization code.

The linkage editor permits an installation to assign the APF-authorization code to load modules either through the PARM field on the link edit step or through a linkage editor control statement.

To assign an authorization code via the JCL parameter, AC=1 should be coded in the PARM field of the EXEC statement as follows:

```
//LKED  EXEC    PGM=HEWL,PARM='AC=1',...
```

If no authorization code is assigned in the linkage editor step, the default is non-authorization. The authorization code for a given output module can be overridden with the SETCODE control statement.

The SETCODE statement is used to establish authorization for a specific output load module. If it is used, it must be placed before the NAME statement for the load module. The format of the SETCODE statement is:

```
SETCODE           AC( 1 )
```

If more than one SETCODE statement is assigned to a given output load module, the last statement found will be used.

In the example in Figure 8, the SETCODE statement assigns an authorization code to the output load module MOD1.

```
//LKED           EXEC       PGM=HEWL
//SYSPRINT       DD         SYSOUT=A
//SYSUT1         DD         UNIT=SYSDA,SPACE=(TRK,(10,5))
//SYSLMOD        DD         DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSLIN         DD         DSNAME=&&LOADSET,DISP=(OLD,PASS),
//                          UNIT=SYSDA
//               DD         *
  SETCODE                   AC(1)
  NAME                      MOD1(R)
/*
```

Figure 8. Assigning Authorization via SETCODE

Note: The authorization code of a load module has meaning only when it resides on an APF-authorized library and when it is executed as the first program of a job step.

APF Summary

The table in Figure 9 summarizes the action taken when an SVC service routine is accessed via an SVC instruction or when a load module is accessed via the LINK, XCTL, LOAD, or ATTACH macro instruction.

Calling Program	Called Program			Results
	SVC Routine	Load Module		
	Restricted	On APF-Authorized Library	Link Edited With APF-Authorized Code	
Authorized*	YES			Successful
	NO			Successful
		NO	NO	306 Abend 1
		NO	YES	306 Abend 1
		YES	NO	Successful 2
		YES	YES	Successful
Not Authorized	YES			047 Abend
	NO			Successful
		NO	NO	Successful
		NO	YES	Successful
		YES	NO	Successful
		YES	YES	Successful

*A program is authorized if it is executing in supervisor state, or with a system key, or as part of an APF-authorized job step.

In these cases, the called program will fail if it executes functions which require authorization.

Figure 9. Authorization Results

Note 1: When the load module is being attached as a job step TCB, the linkage is successful; a 306 abend does not occur. The initiator attaching a job step task is an example of this case.

Note 2: If the load module (the called program) is the first program in the job step task (that is, the linkage is ATTACH of a job step TCB), then the linkage will be successful. However, the program is APF-authorized only if it was link edited as being APF-authorized. For example, if the initiator (an authorized program) attaches a job whose program comes from an APF-authorized library, but where the load module is not link edited as APF-authroized, then the attach is successful but the job runs non-APF authorized.

Changing System Status

The MODESET macro instruction alters selective fields of the program status word (PSW). The standard form of MODESET may be coded in two separate ways: one form generates an SVC and the other form generates inline code.

Generating an SVC

This form of MODESET, executable as APF-authorized, in supervisor state, or under protection key 0-7, changes the status of programs between supervisor state and problem program state, and key zero and non-key zero. The parameters that must be specified to perform the changes are MODE and KEY respectively.

The MODE parameter specifies whether bit 15 of the PSW is to be turned on or off. When bit 15 is one, the CPU is in the problem state. When bit 15 is zero, the CPU is in the supervisor state.

The KEY parameter specifies whether bits 8-11 are to be set to zero or set to the value in the caller's TCB. Bits 8-11 form the CPU protection key. The key is matched against a key in storage whenever information is stored, or whenever information is fetched from a location that is protected against fetching.

Generating Inline Code

This form of MODESET, executable only in supervisor state, is used to ensure that storage areas and the control program functions they are associated with have the same protection key. The EXTKEY parameter of MODESET may be coded to indicate the key to be set in the current PSW.

The following keys may be set:

- Scheduler
- Job entry subsystem
- Real storage management
- Virtual storage management
- System resource management
- Supervisor
- Data management
- Telecommunications access method
- Key of zero
- Key of TCB
- Key of type 1 SVC issuing MODESET
- Key of type 2, 3, or 4 SVC issuing MODESET

Other parameters of MODESET allow the original key to be saved and restored upon completion of the desired changes.

Interprocessor Communications (IPC)

Interprocessor communications is a function in a multiprocessing system that provides communication between CPUs sharing the same control program. Those executing functions which require a CPU or program action on one or more CPUs will use the IPC interface to invoke the desired action. The IPC function uses the signal processor (SIGP) instruction to provide the necessary hardware interface between the CPUs.

Service Classes

The SIGP instruction provides twelve distinct hardware functions to support two classes of IPC services -- *direct* and *remote:*

Direct class - These services are defined for those control program functions which require the modification or sensing of the physical state of one of the configured CPUs. Ten of the twelve SIGP hardware functions are defined as IPC direct services, and are accessible via the DSGNL macro instruction.

Sense: The specified CPU presents its status to the issuing CPU. No other action is caused at the specified CPU.

Start: The specified CPU is placed in the operating state. The CPU does not necessarily enter the operating state during the execution of the SIGP instruction. No action is caused at the specified CPU if that CPU is in the operating state when the order code is accepted.

Stop: The specified CPU stops. The CPU does not necessarily enter the stopped state during the execution of the SIGP instruction. No action is caused at the specified CPU if that CPU is in the stopped state when the order code is accepted.

Restart: The specified CPU restarts. The CPU does not necessarily perform the function during the execution of the SIGP instruction.

Initial Program Reset: The specified CPU performs initial program reset. The execution of the reset does not affect other CPUs and does not affect channels not configured to the CPU being reset. The reset operation is not necessarily completed during the execution of the SIGP instruction.

Program Reset: The specified CPU performs program reset. The execution of the reset does not affect other CPUs and does not affect channels not configured to the CPU being reset. The reset operation is not necessarily completed during the execution of the SIGP instruction.

Stop and Store Status: The specified CPU stops and stores status. The CPU does not necessarily complete the operation, or even enter the stopped state, during the execution of the SIGP instruction.

Initial Microprogram Load: The specified CPU performs initial program reset and then initiates the initial-microprogram-load function. The latter function is the same as that which is performed as part of manual initial micro-program loading. The operation is not necessarily completed during the execution of the SIGP instruction.

Initial CPU Reset: The specified CPU performs initial CPU reset. The execution of the reset does not affect other CPUs and does not cause any channels, including those configured to the specified CPU, to be reset. The reset operation is not necessarily completed during the execution of the SIGP instruction.

CPU Reset: The specified CPU performs CPU reset. The execution of the reset does not affect other CPUs and does not cause any channels, including those configured to the specified CPU, to be reset. The reset operation is not necessarily completed during the execution of the SIGP instruction.

Remote class - These services are defined for those control program functions which require the execution of a software function on one of the configured CPUs. The two remaining SIGP functions are defined as remote services:

External Call: An "external call" external-interruption condition is generated at the specified CPU. The interruption condition becomes pending during the execution of the SIGP instruction. The associated interruption occurs when the CPU is interruptible for that condition. Only one external-call condition can be kept pending in a CPU at a time. The external-call function is accessible via the RPSGNL macro instruction.

Emergency Signal: An "emergency-signal" external-interruption condition is generated at the specified CPU. The interruption condition becomes pending during the execution of the SIGP instruction. The associated interruption occurs when the CPU is interruptible for that condition. At any one time the receiving CPU can keep pending one emergency-signal condition for each CPU of the multiprocessing system, including the receiving CPU itself. The emergency-signal function is accessible via the RISGNL macro instruction.

Status Conditions

Eight status conditions are defined whereby the issuing and specified CPUs can indicate their response to the designated hardware function. The status conditions are contained in register 0 and are:

Equipment Check: This condition exists when the CPU executing an instruction detects equipment malfunctioning that has affected only the execution of the instruction and the associated hardware function. The order code may or may not have been transmitted, and may or may not have been accepted, and the status bits provided by the specified processor may be in error.

External Call Pending: This condition exists when an external-call interruption condition is pending in the specified CPU because of a previously issued SIGP instruction. The condition exists from the time an external-call function is accepted until the resultant external interruption has been completed. The condition may be due to the issuing CPU or another CPU. The condition, when present, is indicated only in response to sense and to external call.

Stopped: This condition exists when the specified CPU is in the stopped state. The condition, when present, is indicated only in response to sense.

Operator Intervening: This condition exists when the specified CPU is executing certain operations initiated from the console or the remote operator control panel. The particular manually initiated operations that cause this condition to be present depend on the model and on the specified functions. This condition, when present, can be indicated in response to all functions. Operator intervening is indicated in response to sense if the condition is present and precludes the acceptance of any of the installed orders. The condition may also be indicated in response to unassigned or uninstalled orders.

Check Stop: This condition exists when the specified CPU is in the check-stop state. The condition, when present, is indicated only in response to sense, external call, emergency signal, start, stop, restart, and stop and store status. The condition may also be indicated in response to unassigned or uninstalled functions.

Not Ready: This condition exists when the specified CPU uses reloadable control storage to perform a function and the required microprogram is not loaded. The not-ready condition may be indicated in response to all functions except IMPL.

Invalid Function: This condition exists during the communications associated with the execution of SIGP when the specified CPU decodes an unassigned or uninstalled function code.

Receiver Check: This condition exists when the specified CPU detects malfunctioning of equipment during the communications associated with the execution of SIGP. When this condition is indicated, the function has not been initiated and, since the malfunction may have affected the generation of the remaining receiver status bits, these bits are not necessarily valid. A machine-check condition may or may not have been generated at the specified CPU.

For more details on the SIGP instruction, see **IBM System/370 Principles of Operation.**

Event Completion

Cross Memory POST

The POST macro instruction is used to signify the completion of an event by one routine to another routine. Usually, the completion of the event is posted in the user's address space.

The authorized user (executing in supervisor state, under protection key 0-7, or APF-authorized) of the POST macro instruction can use the ASCB and ERRET parameters to schedule an SRB to be dispatched in an address space other than his own. If the caller is authorized to specify the ASCB and ERRET parameters, no check is made to determine if the requested address space is the issuing address space. This use of the POST macro instruction is sometimes known as "cross-memory post."

| The ERRET routine is given control in the user's address space when the error condition is detected. It will receive control enabled, unlocked, in SRB mode, and with the following register contents:

register	**contents**
0	**ECB address**
1	**ASCB address**
2	**original completion code**
3	**completion code from failing address space**
4-13	**unpredictable**
14	**return address**
15	**ERRET address**

The ERRET routine must return control unlocked and enabled.

If cross-memory post is being used, a synchronization problem arises when it becomes necessary to eliminate an ECB which is a potential target for a cross-memory post request. To ensure that all outstanding cross-memory post requests for the ECB have completed, the user must invoke the SPOST macro instruction. (The ECB may or may not be posted, depending on existing conditions. See **OS/VS2 Scheduler and Supervisor Logic** for details.) Since SPOST invokes the PURGEDQ SVC, see the description of PURGEDQ for the restrictions on its use.

Bypassing the POST Routine

The problem programmer may bypass the POST routine whenever the corresponding WAIT has not yet been issued. To do this, it is necessary to issue a TEST UNDER MASK (TM) instruction to determine if the wait bit in the ECB is on. If the wait bit is on, the normal POST routine must be executed. If the wait bit is not on, a COMPARE AND SWAP (CS) of the ECB to the posted condition should be issued. At this time, if the wait bit is on, the normal POST routine must be executed; if the wait bit is not on, the CS will, in effect, post the completion of the event.

Figure 10 demonstrates an example of this use of POST.

```
             L        RX,ECB            Get contents of ECB.
             L        RY,='40000000'    Completion bit and code to be
                                        compared and swapped
CSLOOP       TM       ECB,X'80'         Wait bit on?
             BO       DOPOST            If yes, then execute post.
             CS       RX,RY,ECB         Compare and swap completion bit
                                        and code.
             BZ       POSTDONE          Branch if CS is successful.
DOPOST       POST     ECB
POSTDONE     EQU      X
```

Figure 10. Bypassing the POST Routine

Waiting for Event Completion

The EVENTS macro instruction allows a user to wait for the completion of one of a series of events and be directly informed by the system which of the events have completed. Branch entry to this function, significantly more efficient than SVC entry, is available to users executing in key 0, supervisor state, and holding only the local lock.

Branch entry is accomplished by specifying BRANCH=YES on the EVENTS macro instruction. If this parameter is used, the branch entry routine will perform all normal WAIT processing and ECB initialization. BRANCH=YES may be specified in conjunction with either WAIT=YES, WAIT=NO, or ECB=.

- If WAIT=YES is specified, control will later be returned to the dispatcher, even though there may be ECBs posted to the EVENTS table. The local lock will be freed by EVENTS. Before the EVENTS macro instruction with the WAIT=YES option is specified, the caller is responsible for establishing the return environment (the PSW and registers in the RB and TCB). EVENTS will store a pointer to the first completed EVENTS entry into the TCB register 1 save location. (This service is not available to Type 1 SVCs or SRBs.)
- If WAIT=YES is not specified, control will later be returned to the caller. The local lock will not be freed by EVENTS.

System Integrity

System integrity is defined as the ability of the system to protect itself against unauthorized user access to the extent that security controls cannot be compromised. That is, there is no way for an unauthorized problem program using any system interface to bypass store or fetch protection, bypass password checking, or obtain control in an authorized state.

Note: An authorized program in MVS is one that executes in a system key (keys 0-7), in supervisor state, or is authorized via the Authorized Program Facility (APF).

Documentation on System Integrity

This section contains information about MVS system integrity. The related topic of security in regard to the physical environment of a computing system is discussed in **Data Security and Data Processing**, GC20-1370 through GC20-1375.

Restricted functions (those that can be requested only by authorized programs) are documented separately from non-restricted functions in **OS/VS2 System Programming Library: Data Management**, GC26-3830.

These publications describe the restricted functions and the means of authorizing appropriate programs to use those functions. Macro instructions that require the issuing program to be authorized are listed in the contents and described in the text of these books. Review the contents of these books to determine what functions require authorization.

Installation Responsibility

To ensure that system integrity is effective and to avoid compromising any of the integrity controls provided in the system, the installation must assume responsibility for the following:

- Physical environment of the computing system.
- Adoption of certain procedures (for example, the password protection of appropriate system data sets) that are a necessary complement to the integrity support within the operating system itself.
- That its own modifications and additions to the control program do not introduce any integrity exposures. That is, all installation-written authorized code (for example, an installation SVC must perform the same or an equivalent type of validity checking and control that the MVS control program employs to maintain system integrity.

Elimination of Potential Integrity Exposures

MVS system integrity support restricts only unauthorized problem programs. It is the responsibility of the installation, to verify that any authorized programs added to the system control program will not introduce any integrity exposures. To do this effectively, an installation should consider these areas for potential integrity exposure:

- User-supplied addresses for user storage areas.
- User-supplied addresses for protected control blocks.
- Resource identification.
- SVC routines calling SVC routines.
- Control program and user data accessibility.
- Resource serialization. (See the section "Locking".)

Each of the following descriptions is a guideline to aid the installation in:

- Eliminating that area as a potential integrity exposure.
- Determining whether an impact on existing installation-written code might occur, especially where that code is dependent on the use of non-standard interfaces to the system control program.

There should be no impact on installation-written routines that use standard interfaces (problem program/system interface described in an SRL) becuase no standard interfaces for system integrity support have been removed from the MVS system control program. However, some routines now require authorization for use.

User-Supplied Addresses for User Storage Areas

A potential integrity exposure exists whenever a routine having a system protection key (key 0-7) accepts a user-supplied address of an area to which a store or fetch is to be done. If the system routine does not adequately validate the user-supplied address to ensure that it is the address of an area accessible to the user for storing and fetch data, an integrity violation can occur when the system-key routine:

- Stores into (overlays) system code or data (for example, in the nucleus or the system queue area), or into another user's code or data.
- Moves data from a fetch-protected area that is not accessible to the user (for example, fetch-protected portion of the common service areas) to an area that is accessible to the user.

The elimination of this problem requires that system-key routines always verify that the entire area to be stored into, or fetched from, is accessible (for storing or fetching) to the user in question. The primary validation technique is the generally established MVS convention that system-key routines obtain the protection key of the user before accessing the user-specified area of storage. For example, MVS data management SVC routines (which generally execute in key 5) assume the user's key before modifying a data control block (DCB) or an I/O block (IOB).

User-Supplied Addresses for Protected Control Blocks

A potential integrity exposure exists whenever the control program (system key/privileged mode) accepts the address of a protected system control block from the user. For most system control blocks this situation should not be permitted to exist. However, in certain cases it is necessary to allow the user to provide the address of a system control block that describes his allocation/access to a particular resource (for example, a data set), in order to identify that resource from a group of similar resources (for example, a user may have many data sets allocated). Inadequate validity checking in this situation can create an integrity exposure, since an unauthorized problem program could provide its own (counterfeit) control block in place of the system block and thereby gain the ability to:

- Access a resource in an uncontrolled manner (since the control block in this case would normally define the restrictions, such as read-only for a data set, on the user's allocation to the resource).
- Gain control in a privileged state (because such control blocks might contain the addresses of routines that run in privileged mode or with a system (0-7) key).
- Cause various other problems depending on exactly what data is in the control block involved.

To avoid this type of exposure, the control program must verify, for every such address accepted from a problem program, that the address is that of:

1. A protected control block created by the control program.
2. The correct type of control program block (for example, a TCB versus a DEB, or a QSAM DEB versus an ISAM DEB).
3. A control block created for use in connection with the user (job step) that supplied the address.

In MVS, verification is generally accomplished by establishing a chain or table of the particular type of control block to be validated. This chain or table is located via a protected and jobstep-related control block that is known to be valid. Addresses that are not allowed to be supplied by the user, will be located via a chain of protected control blocks that begins with a control block known to be valid or fixed at a known location at IPL time, such as the CVT. Therefore, a control block can only be entered in the chain/table by:

- An authorized program satisfying point 1.
- Definition, where the chain/table establishes the type of control block satisfying point 2.
- Definition, where each chain/table is located only through a jobstep-related control block satisfying point 3.

Note: This does not imply that a system routine must go back to the CVT or similar control block every time it wants to establish a valid chain. Typically, a control block address not too far down on such a chain is available already validated in a register. For example, the first load of an SVC can receive control with a valid TCB address in a register.

Resource Identification

Resource identification is another area that can be subject to integrity exposures. Exposures can result if the control program does not maintain and use sufficient data to uniquely distinquish one resource from other similar resources. For example, a program must be identified by both name and library to distinguish it from other programs. The consequences of inadequate resource identification are problems such as the ability of an unauthorized problem program to create counterfeit control program code or data, or to cause varying types of integrity problems by intermixing incompatible pieces of control program code and/or data.

The genrral solution can only be stated as the reverse of the problem; that is, the control program must maintain and use sufficient (protected) data on any control program resource, to distinquish between that resource and other control program or user resources. The following are examples of the controls that MVS employs to comply with the requirement:

- In general, authorized program requests to load other authorized programs are satisfied only from authorized system libraries (see the topic "Control Program Extensions" described in this section.)
- MVS takes explicit steps to ensure that routines loaded from authorized system libraries are used only for their intended purpose. This includes expanded validity checking to remove any potential for the unauthorized program to specify explicitly which of the authorized library routines are to gain control in any given situation.
- Sensitive system control blocks are validated as being the "correct" blocks to be used in any given control program operation. (See the topic "User-Supplied Addresses of Protected Control Blocks " described in this section.)

SVC Routines Calling SVC Routines

A potential problem area exists whenever a problem program is allowed to use one SVC routine (routine A) to invoke a second SVC routine (routine B) that the problem program could have invoked directly. An integrity exposure occurs if:

- SVC routine B bypasses some or all validity checking based on the fact that it was called by SVC routine A (an authorized program) or
- User-supplied data passed to routine B by routine A either is not validity checked by routine A, or is exposed to user modification after it was validated by routine A.

These problems will not exist if the user calls SVC routine B directly, because the validity checking will be performed on the basis of the caller being an unauthorized program.

SVC routine A, which is aware that it has been called by an unauthorized program, must ensure that the proper validity checking, etc., is accomplished. However, it is usually not practical for SVC routine A to do the validity checking itself, because of the potential for user modification of the data prior to or during its use by SVC routine B. The general solution should be for SVC routine A to provide an interface to SVC routine B, informing routine B that the operation is being requested with user-supplied data in behalf of an unauthorized problem program (implying that normal validity checking should be performed).

In practice, in MVS, most SVC B-type routines that could be subject to this problem use the key of their caller as a basis for determining whether or not to perform validity checking. Therefore, most SVC A-type MVS routines have simply adopted the convention of assuming the key of their caller before calling the SVC B routine. (For additional information see the section "Writing SVC Routines" later in this book.)

Control Program and User Data Accessibility

Important in maintaining system integrity is the consideration of what system data is sensitive and must be protected from the user, and what data can be exposed to user manipulation. The implications of the expôsure of the wrong type of data are obvious.

In general, it is necessary to store protect the following types of data:

- Code, and the location of code, that is to receive control in an authorized state.
- Work areas for such code, including areas where it saves the contents of registers.
- Control blocks that represent the allocation or use of system resources.

MVS maintains such items in system storage, or in a separate address space in the case of some APF-authorized programs.

It may also be necessary to protect, for a limited period, certain data that is normally under the control of the user (for example, to prevent its modification during a critical operation). In this case MVS provides fetch protection for such data if:

- The data consists of proprietary information (such as passwords).
- The control program cannot determine the nature of the contents of the data area.

Control Program Extensions

This potential problem area involves the somewhat hazy distinction that exists between the control program and certain types of problem programs. In most installations, there are problem state/user key (keys 8-15) programs that are actually extensions to the control program in that they are allowed (by means of various special SVCs, etc.) to bypass normal system controls over access to system resources. For example, a special utility program that scans all the data on a pack might be able to avoid the normal system extent checking on a direct access volume.

If an installation has its own control program extensions and SVCs that allow the bypass of normal system security or integrity checks (for example, an SVC that returned control in key 0), and if such SVCs are not currently restricted from use by an unauthorized program, thee APF facility should be used to restrict them and to authorize the control program extensions that use them.

Installation personnel should understand the distinction between an authorized program and an authorized user. A program that utilizes a restricted function must be authorized. Any user, however, can submit a job to execute an authorized program. Any program that must be restricted to an individual or a class of users must be restricted by means of existing data set security facilities. The program must be placed placed on a separate password-protected library other than LINKLIB, SVCLIB, or LPALIB. IEH utilities might be placed on a separate password-protected library to limit their use to system programmers.

Interruption, Recovery/Termination, and Dumping Services

The supervisor offers many services to assist in the detection and processing of abnormal conditions during the execution of the system. Certain types of abnormal conditions are detected by the hardware and cause program interruptions to occur (for example, if an attempt is made to execute an instruction with an invalid operation code). Other abnormal conditions are detected by the software (for example, an attempt to open a data set which is not defined to the system causes an ABEND to be issued by the Open routine).

The supervisor provides facilities for writing exit routines to handle specific types of interruptions and abnormal conditions. The supervisor initiates the recovery/termination process of your program when you request it by issuing an ABEND macro instruction, or when the control program detects a condition that will degrade the system or destroy data.

The services discussed in this chapter include:

- program interruption processing SPIE macro instruction)
- recovery/termination (CALLRTM macro instruction)
- functional recovery routines (SETFRR macro instruction)
- task recovery (STAE and ESTAE macro instructions)
- installation-written clean-up routines
- virtual storage dumping (SDUMP macro instruction and CHNGDUMP command)

SPIE Processing

The SPIE macro instruction provides a problem program with a means of specifying an error exit routine in response to one or more program error interruptions. SPIE and its related services are discussed in detail in **OS/VS2 Supervisor Services and Macro Instructions.**

For the problem programmer, interruptions 1-15 may be specified in the SPIE macro instruction. For the installation-authorized system programmer, interruption 17 may also be specified. Interruption 17 designates page faults and may be specified so that a user-written SPIE routine will get control on a page fault before any supervisor routine. The user-provided SPIE routine only gets control in problem program state on a page fault for the program issuing the SPIE 17. The SPIE 17 routine covers page faults at the task level and any RBs executing under the task for which the SPIE was issued. Since the SPIE 17 user may not be in supervisor state, the routine will generally not be able to resolve the page fault in a manner compatible with the operation of the system. Hence, the SPIE 17 routine will usually resolve the page fault by invoking PGLOAD/PGFIX functions of the paging supervisor.

The SPIE 17 routine will get control in problem program state. If the program is in supervisor state at the time the SPIE 17 is issued, it will be abnormally terminated with a 30E abend completion code. If the program is in supervisor state and takes a page fault while the SPIE 17 routine is active, the SPIE 17 exit routine will *not* get control. Supervisor routines will resolve the page fault and continue program processing without abending the program.

During SPIE processing, the Program Check First Level Interrupt Handler (FLIH) will pass control directly to the SPIE routine after some set-up processing via a LPSW. The Program Check FLIH sets a recursion indicator to cover any PIE/PICA references during the setup processing done to handle a page fault incurred in the problem program. If a page fault occurs on a reference to the PIE/PICA, the Paging Supervisor will be given control to handle the original page fault. Processing will continue in the problem program once the page fault is resolved. The SPIE exit routine does not receive control at all since the FLIH is not able to obtain the information needed by the SPIE routine as input parameters.

If interruption 17 is to be used, the programmer must page fix the PIE, PICA, and SPIE exit programs and data areas. The SPIE exit routine must be aware that page faults can occur after issuing the SPIE macro instruction for interruption 17 and prior to fixing the required control blocks. If the page fault occurs at this time, the Program FLIH will try to pass control to the SPIE 17 routine after the setup processing mentioned. If the PIE/PICA can be validly referenced, control will be passed to the SPIE 17 routine. If the SPIE 17 routine page faults, the page fault will be resolved by the Paging Supervisor if the routine is not running disabled. A disabled page fault will cause a 0C4 abend. Once the page fault is resolved, normal processing will continue in the SPIE 17 routine.

It is important to note that the SPIE 17 routine may not get control on every page fault due to the recursion logic mentioned above.

Recovery/Termination

The recovery/termination manager (RTM) monitors the flow of control of software recovery processing and supplies the services of normal and abnormal task and address space termination. Its purpose is to select the appropriate recovery or termination process according to the status of the system and the request of its invokers.

The RTM may be called to perform its recovery and termination services on behalf of the caller or to direct its services to another routine. It is invoked by two macros: the ABEND macro and the CALLRTM macro.

The recovery/termination process is invoked for the following events:

- Unanticipated program checks - Program checks protected by SPIE routines are not handled by the RTM.
- Machine checks.
- Invalid issuance of an SVC while locked, disabled, or in SRB mode.
- I/O error on page-in request.
- Restart Key - An operator-initiated recovery action, requested by depressing the console RESTART key.
- Request by an authorized caller to terminate a task or a memory.
- ABEND macro - This may be system-issued or user-issued.

When one of these events occurs, the RTM is given control. If a recovery exit has been specified (via SETFRR, STAE, or ESTAE) it is invoked to recover or clean up for the process in control. Should this recovery routine be unable to recover from the incident (request termination or fail itself) the previously established recovery exit will be invoked. This process is called percolation. In the event that all recovery routines are unable to recover, the process is terminated.

The recovery routines are given control in LIFO (last in, first out) order. If all recovery routines established via SETFRR percolate, the related task is abended, if one exists. Then the STAE/ESTAE routines which were created by the task are invoked.

Invoking the Recovery/Termination Manager

CALLRTM

A routine should use CALLRTM to direct the Recovery/Termination services to a task or routine other than itself or its ancestors (callers). CALLRTM may be issued only by key 0 priviledged routines. Control is returned to the issuer of the macro if the TYPE=ABTERM or TYPE=MEMTERM options are coded.

The following locking, workarea, and special considerations should be noted when using CALLRTM.

TYPE=ABTERM

If TYPE=ABTERM is specified in the CALLRTM macro instruction, the RTM processing is directed towards another task. In this situation, the following locking and save area requirements should be considered:

- If the TCB parameter is specified as 0 (or defaulted to 0), and the ASID parameter is omitted, the current task in the current address space is abnormally terminated. In this situation, the caller must be disabled (for example, hold any of the spin locks) and need not pass a save area via register 13. If dump options are supplied, they must be contained in fixed pages.
- If the TCB parameter is specified as an address, and the ASID parameter is omitted, the task associated with the specified TCB in the current address space is abnormally terminated. In this situation, the caller must own the local lock, and a save area is not required.
- If the ASID parameter is specified, the ABTERM function is scheduled as a service request block SRB. Although there is no specific lock requirement, the caller must pass the address of an 18-word work area in register 13. In this way, the ASID parameter allows processing across address spaces and allows processing in the current address space for routines that cannot acquire the proper locks (possibly because of hierarchy conflicts).

TYPE=MEMTERM

If TYPE=MEMTERM is specified in the CALLRTM macro instruction, the RTM processing is directed towards an address space. In this situation, the following information should be considered:

- If the ASID parameter is specified as nonzero, the specified address space is abnormally terminated. The caller need not be disabled or own any locks. The caller must pass the address of an 18-word work area in register 13.
- If the ASID parameter is specified as 0 or is omitted, the current address space is abnormally terminated. The caller need not be disabled or own any locks. The caller must pass the address of an 18-word work area in register 13.

Since the MEMTERM process circumvents all task recovery and task resource manager processing, its use is restricted to a select group of routines which can determine that task recovery and task resource manager clean-up is either not warranted or will not successfully operate in the address space being terminated. These routines include:

- Paging supervisor, when it determines that it cannot swap in the LSQA for an address space.
- Memory create, when it determines that an address space cannot be initialized.
- RTM or supervisor control FRR, when it determines that uncorrectable translation errors are occuring in the address space.
- RTM, when it determines that task recovery and termination cannot take place in the current address space.
- Region control task, when it has determined that the address space may become permanently deadlocked--that is, unusable--or the status of the address space is unpredictable due to an error during swap-out processing.
- RTM, when all tasks in the address space have terminated.

ABEND

The ABEND macro instruction should be used by any routine, including supervisor state, locked, disabled or SRB routines, to request the services of the RTM to be directed to itself (cause entry into its recovery routine) or to its callers. The issuer of ABEND should remove its own recovery exit if it wishes its caller to receive the services of the RTM. Control is never returned to the issuer of the macro (except by using the STA/ESTA/FRR retry mechanism). See OS/VS2 **Supervisor Services and Macro Instructions** for a description of the ABEND macro.

Types of Recovery Routines

Functional Recovery Routines (FRRs)

FRRs are recovery routines established to protect locked, disabled, or SRB mode routines. An FRR is identified to the RTM by coding the SETFRR macro instruction. When a functional recovery routine is invoked it will run in the state of the system (enabled or disabled) and with the locks that were held at the time of the error, or as modified by previous FRRs.

Task Recovery Routines (STAE/STAI ESTAE/ESTAI)

Task recovery is accomplished through STAE/STAI or ESTAE/ESTAI routines. Issuance of the STAE or ESTAE macro instruction, or the ATTACH macro instruction with the STAI or ESTAI parameter, allows the user to intercept a scheduled ABEND. Control is given to a user-specified exit routine in which the user may diagnose the cause of the ABEND, and specify perform pre-termination processing, or specify a retry address if he wishes to prevent the termination.

Establishing Recovery Routines

Functional Recovery Routines

The SETFRR macro instruction provides control program functions with the ability to define their recovery in the FRR LIFO stack which is used during system recovery management. The LIFO stack is maintained by the recovery termination manager and contains the addresses of the FRRs established to protect a single path through supervisor control and SRB code.

The issuer of SETFRR must be key 0 as the stack is maintained in protected storage. Furthermore, the SETFRR issuer must be disabled, locked, or in SRB mode (and therefore supervisor state) to maintain FRR stack integrity. The FRR stacks are serialized by disablement or ownership of a global spin lock. For owners of suspend locks (CMS or LOCAL) and SRB functions, the stack is saved and restored as part of the paths operating environment by the supervisor control functions on interruptions and redispatch.

If RTM is invoked, the last FRR established is given control and executes in the system mode (locks held, disablement, SRB mode) at the time of the error.

The FRR must indicate to the RTM the action to be performed. This is done via settings in the system diagnostic work area (SDWA) which is used for communication between FRRs and the RTM. If the FRR requests percolation (no retry) the previously established FRR will be given control. Each FRR is given control in LIFO order until retry is requested or the stack is exhausted.

When FRR processing is exhausted, the RTM determines if a task should be terminated (current or related task if in SRB mode). The RTM then sets the task up for ABEND and task recovery will take place if task recovery exits exist, otherwise, the task will be abnormally terminated.

An FRR environment is canceled when a routine issues SETFRR with the delete option, when that FRR requests percolcation or when the system becomes enabled task mode and an interruption and redispatch occurs. The FRR entry should be deleted before the function returns to its caller, otherwise, the FRR may get control for its caller's error.

All SETFRR users must include the DSECTs for the FRR stack (via the IHAFRRS mapping macro instruction) and the PSA (via the IHAPSA mapping macro instruction) prior to using the SETFRR macro instruction. In addition, all disabled, locked, and SRB routines which define recovery must be key 0 supervisor state when using the SETFRR macro instruction. It is necessary to copy IHAPSA from MODGEN into MACLIB.

Note: The size of all FRR stacks satisfies the recovery needs of the control program. If additional FRRs are placed on the stack, thereby causing the size to be exceeded, the routine issuing the SETFRR macro instruction will be abnormally terminated. Any user-written routines outside the control program may add one, and only one, FRR to the stack; if more than one is added, abnormal termination may occur. This applies to all of the recovery stacks, including the normal stack. The normal FRR stack is used by control program routines which are invoked on behalf of the user.

Task Recovery Routines

STAE/STAI Exit Routines: The STAE macro instruction causes a recovery routine address to be made known to the control program. This recovery routine is associated with the task and the RB which issued STAE. Use of the STAI option on the ATTACH macro instruction also causes a recovery routine to be made known to the control program, but the routine is associated with the subtask created via ATTACH. Furthermore, STAI recovery routines are propagated to all lower-level subtasks of the subtask created with ATTACH that specified the STAI parameter.

If a task is scheduled for abnormal termination, the exit routine specified by the most recently issued STAE macro instruction is given control and executes under a program request block created by the SYNCH service routine. Only one STAE exit routine will receive control. The STAE exit routine must specify, by a return code in register 15, whether a retry routine is to be scheduled. If no retry routine is to be scheduled (return code = 0) and this is a subtask with STAI recovery routines, the STAI recovery routine is given control. If there is no STAI recovery routine, abnormal termination continues.

If there is more than one STAI recovery routine existing for a task, the newest one receives control first. If it requests that termination continue (return code = 0), the next STAI routine will receive control. This will continue until either all STAI routines have received control and requested that the termination continue, a STAI routine requests retry (return code = 4 or 12), or a STAI routine requests that the termination continue but no further STAI exits receive control (return code = 16).

Programs running under a single TCB may issue more than one STAE macro instruction with the create (CT) parameter. Each issuance makes the previous STAE environment temporarily inactive. The environment will become active when the current STAE environment is canceled.

A STAE environment is canceled when the RB which created it goes away (unless it issues XCTL and specified the XCTL=YES parameter on the STAE macro instruction), when the STAE macro instruction is issued with the CANCEL option, or when the STAE routine receives control. If a STAE exit routine receives control and requests retry, the retry routine will have to reissue the STAE macro instruction if it wants continued STAE protection.

A STAI environment is canceled if the task completes or if it requests that termination continue and no further STAI processing be done. In the later case, all STAI exits for the task are canceled.

ESTAE/ESTAI Exit Routines: The ESTAE macro instruction, like the STAE macro instruction, causes a task and RB-related recovery environment to be created. Use of the ESTAI option on the ATTACH macro instruction also identifies a recovery routine to the control program, but the routine is associated with the subtask created via ATTACH. Furthermore, ESTAI recovery routines are propagated to all lower-level subtasks of the subtask created with the ATTACH that specified the ESTAI parameter. (See Figure 11.)

If a task is scheduled for abnormal termination, the recovery routine specified by the most recently issued ESTAE macro instruction is given control and executes under a program request block created by the SYNCH service routine. On return, the exit routine may indicate whether a retry routine should be scheduled or whether termination should continue. If it requests that termination continue, the next ESTAE routine for the task receives control. If all ESTAE routines request that termination continue, or if none exist, the ESTAI routines, if any, receive control.

Before the initial recovery routine receives control, the purge and asynchronous processing requests specified when the exit was created are performed by the control program. The I/O processing requested will be performed only for the first exit routine selected. Subsequent routines will receive an indication of the I/O processing previously performed, but no additional I/O processing will be performed. The asynchronous processing request, however, will be performed for each routine.

Each ESTAE exit established by a task is eligible to receive control. If an error occurs, the most recently created ESTAE will be entered. If it requests that termination continue, or it fails itself, the next ESTAE exit, if any, will be entered. This will continue until an ESTAE exit requests retry or all exits for the task are exhausted.

Both STAE and ESTAE exits can exist for the same task. However, only one STAE exit will receive control. STAI and ESTAI exits will receive control after all ESTAE exits and one STAE exit, if any, have been processed.

An ESTAE environment is canceled when the RB which creates it goes away (unless it issues an XCTL and specified the XCTL=YES parameter on the ESTAE macro instruction), when the ESTAE macro instruction is issued with the CANCEL option, the exit routine fails, or the exit routine requests that termination continue.

Figure 11 demonstrates the queuing structure of the ESTAE routines and the propagation of ESTAI to subtasks.

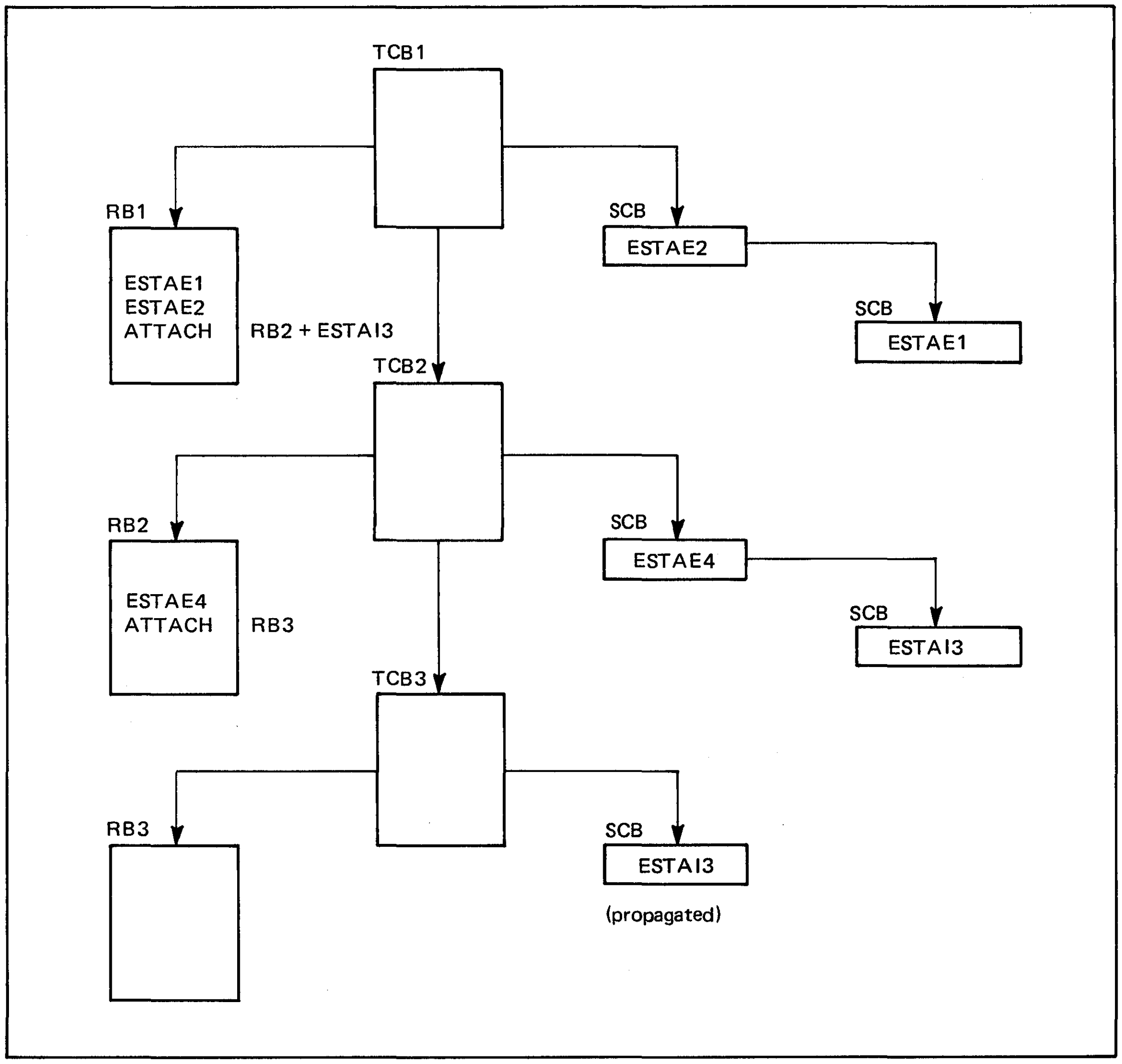


Figure 11. ESTAE Environment

RTM/Recovery Routine Interface

Interface to Functional Recovery Routines

Prior to giving control to FRRs the RTM locates and initializes a work area which contains information about the error. This work area is called the system diagnostic work area (SDWA) and is 512 bytes long. The first word of the SDWA contains the address of the six-word parameter area returned by the SETFRR macro instruction when the PARMAD keyword is specified on the SETFRR macro instruction.

FRRs represented on the system recovery stack receive control via the LPSW instruction. The most recent FRR address on the stack is merged with the saved error PSW so that the FRR will get control in the system state representing the CPU status at the time of error. Note: The PSW will always be enabled for machine checks and DAT.

Upon entry to the FRR, parameter registers are as follows:

Register	0	Address of a 200-byte FRR work area.
Register	1	Address of the SDWA.
Register	14	Return address.
Register	15	Address of the FRR.

Note that register 13 is not part of the FRR interface. Any register may be used without saving it, but caution should be used to maintain the return address supplied in register 14.

The locks held and disablement will be the same as at the time of error, except for percolated-to FRRs in which case, lock status may change if previous exits requested that locks be freed.

The SDWA contains information pertaining to the error (that is, registers and PSW). The SDWAFMID field contains an indication of the memory in which an error occured if recovery is being initiated in another memory. If this field is not zero, no reference may be made to private (local) area.

An FRR should use the lock freeing capabilities of the RTM (via SETRP) to free locks obtained by the mainline, if that is the action desired. In any event, an FRR must not free the last global lock causing enablement, or the local lock. The RTM must be used in these cases. If the RTM is used to free all locks required, the above checks can be avoided.

RTM freeing of locks will only be honored on percolation. Freeing or obtaining locks for or in retry situations must be done by the retry routine.

When the FRR has completed, it should use the SETRP macro instruction to inform RTM of the action it desires. This macro instruction will initialize the SDWA with these options.

The SETRP macro instruction is described in **Supervisor Services and Macro Instructions**, with the exception of several restricted parameters which are described in this publication.

Interface to Task Recovery Routines

Interface to a STAE/STAI Exit: Prior to going to a STAE/STAI recovery routine, the control program attempts to obtain and initialize a work area which contains information about the error. This work area is called the system diagnostic work area (SDWA), and is 512 bytes long. The first word of the SDWA contains the address of the parameter list specified on the STAE macro instruction or the STAI parameter or the ATTACH macro instruction.

Upon entry to the STAE routine, parameter registers are as follows:

If an SDWA was obtained:

register	0	a code indicating the type of I/O processing performed:	
		0	active I/O has been quiesced and is restorable.
		4	active I/O has been halted and is not restorable.
		8	no active I/O at ABEND time.
		16	active I/O, if any, was allowed to continue.
register	1	address of the SDWA.	
register	13	save area address.	
register	14	return address.	
register	15	address of STAE exit routine.	

If no SDWA was available:

register	0	code 12 to indicate that no SDWA was obtained.
register	1	ABEND completion code.
register	2	address of user-supplied parameter list.
register	13	unpredictable.
register	14	return address.
register	15	address of STAE exit routine.

When the STAE or STAI routine has completed, it should return to the control program via the contents of register 14. Register 15 should contain one of the following return codes:

return code	action
0	Continue the termination. The next STAI, ESTAI, or ESTAE exit will be given control. No other STAE exits will receive control.
4,8,12	A retry routine is to be scheduled.
16	No further STAI/ESTAI processing is to occur. This code may only be issued by a STAI/ESTAI exit.

For the following situations, STAE/STAIexits will not be entered:

- If the abnormal termination is caused by an operator's CANCEL, job step timer expiration, or the detaching of an incomplete task.
- If the failing task has been in a wait state for more than 30 minutes.
- If the STAE macro instruction was issued by a subtask and the attaching task abnormally terminates.
- If the recovery routine was specified for a subtask, via the STAI parameter of the ATTACH macro instruction, and the attaching task abnormally terminates.
- If a problem other than those above arises while the control program is preparing to give control to the STAE routine.

Interface to an ESTAE/ESTAI Exit: Prior to going to an ESTAE/ESTAI recovery routine, the control program attempts to obtain and initialize a work area which contains information about the error. This work area is called the system diagnostic work area (SDWA) and is 512 bytes long. The first word of the SDWA contains the address of the parameter list specified on the ESTAE macro instruction or the ESTAI parameter of the ATTACH macro instruction.

Upon entry to the ESTAE exit routine, parameter registers are as follows:

If an SDWA was obtained:

register	0	a code indicating the type of I/O processing performed:
		0 active I/O has been quiesced and is restorable.
		4 active I/O has been halted and is not restorable.
		8 no active I/O at ABEND time.
		16 no I/O processing was performed.
register	1	address of the SDWA.
register	13	save area address.
register	14	return address.
register	15	entry point address.

If no SDWA was available:

register	0	code 12 to indicate that no SDWA was obtained.
register	1	ABEND completion code.
register	2	address of user-supplied parameter list.
register	13	unpredictable.
register	14	return address.
register	15	entry point address.

When the ESTAE/ESTAI routine has completed, it should use the SETRP macro instruction to inform the control program of the actions it desires. This macro instruction will initialize the SDWA with these options.

If a work area could not be provided by the control program, a register save area will not be provided either. If no SDWA is available, register 14 must be saved and used as the return register to the control program.

When the ESTAE or ESTAI routine has completed its processing, it should return to the control program via the contents of register 14. Register 15 should contain one of the following return codes if an SDWA was not obtained:

return code	action
0	Continue the termination. Any ESTAE exists established prior to this one will receive control.
4	A retry routine is to be scheduled, and its address is placed in register 0.
16	Termination should be continued. No further ESTAE/ESTAI processing should be performed.

When an ESTAE routine requests retry, the RB queue is terminated up to, but not including, the RB of the program that issued the ESTAE macro instruction. This is done by pointing the RB old PSW to an SVC 3 instruction. In addition, open DCBs which can be associated with the purged RBs are closed and queued I/O requests associated with these DCBs being closed are deleted from the I/O restore chain.

The RB queue purge is an attempt to cancel the effects of partially executed programs that are at a lower level in the program hierarchy than the program under which the retry will occur. However, certain effects on the system will not be canceled by this RB purge. Example of these effects are as follows:

- subtasks created by a program to be purged
- resources allocated by the ENQ macro instruction
- DCBs that exist in dynamically acquired virtual storage

If there are quiesced restorable input/output operations, they can be restored in the ESTAE retry routine by using word 2 in the SDWA. Word 2 contains the pointer to the purged I/O request list (PIRL) passed as a parameter to SVC Restore. SVC Restore is used to have the system restore all I/O requests on the PIRL.

RTM/Retry Routine Interface

FRR-Requested Retry Routine

If an FRR requests a retry routine be given control, by specifying a return code of 4 on SETRP, the following interface will be established:

- Registers 0 - 14 are the registers at the time of error with the exception of registers that the FRR requested be changed.
- Locks held and disablement are the same as on exit from the FRR.
- Protect key will be 0.
- Register 15 will contain the retry routine address. Entry to the retry routine is done via a branch to register 13.

Task Recovery Retry Routines

STAE/STAI Retry Routines: If the STAE retry routine is scheduled, the system automatically cancels the active STAE environment; the preceding STAE environment, if one exists, then becomes the active one. Users wanting to maintain STAE protection during retry must reestablish an active STAE environment within the retry routine, or must issue multiple STAE requests prior to the time that the retry routine gains control.

Like the STAE/STAI exit routine, the STAE/STAI retry routine must be in storage when the exit routine determines that retry is to be attempted. If not already resident in your program, the retry routine may be brought into storage via the LOAD macro instruction by either the user's program or exit routine.

If the STAE/STAI routine indicates that a retry routine has been provided (return code = 4, 8, or 12), register 0 must contain the address of the retry routine. The STAE environment that requested retry is canceled and the request block queue is purged up to, but not including, the RB of the program that issued the STAE macro instruction. This is done by pointing the RB old PSW to an SVC 3 (EXIT) instruction. In addition, open DCBs which can be associated with the purged RBs are closed and queued I/O requests associated with the DCBs being closed are deleted from the I/O restore chain.

The RB purge is an attempt to cancel the effects of partially executed programs that are at a lower level in the program hierarchy than the program under which the retry will occur. However, certain effects on the system will not be canceled by this RB purge. Example of these effects are as follows:

- Subtask created by a program to be purged.
- Resources allocated by the ENQ macro instructions.
- DCBs that exist in dynamically acquired virtual storage.

If there are quiesced restorable input/output operations, they can be restored, in the STAE retry routine, by using word 2 in the SDWA. Word 2 contains the pointer to the purged I/O request list (PIRL) passed as a parameter to SVC Restore. SVC Restore is used to have the system restore all I/O requests on the PIRL. (For additional information on SVC Restore, see **OS/VS2 System Programming Library: Data Management.)**

If an SDWA was obtained, upon entry to the STAE/STAI retry routine, register contents are as follows:

Register	0	0
Register	1	Address of the SDWA.
Register	2-13	Unpredictable.
Register	14	Address of an SVC 3 instruction.
Register	15	Address of the STAE/STAI retry routine.

The retry routine should use the FREEMAIN macro instruction to free the 512 bytes of storage occupied by the work area when the storage is no longer needed. This storage should be freed from subpool 0 which is the default subpool for the FREEMAIN macro instruction.

If the ABEND/STAE interface routine was not able to obtain storage for the work area, register contents are as follows:

Register	0	12
Register	1	ABEND completion code.
Register	2	Address of the PIRL; or 0 if I/O is not restorable.

ESTAE/ESTAI Retry Routines: If the ESTAE/ESTAI routine is specified, the following actions will be performed by the control program prior to scheduling it:

- A dump will be given if requested.
- FREEMAIN of SDWA will be performed, if requested.
- Registers will be updated with user-supplied values, if requested.
- The RB queue will be purged up to the level of the retrying RB.

Outstanding WTORs are not purged prior to scheduling an ESTAE routine.

Retry routines run at the RB associated with the requestor of the ESTAE exit routine causing the retry. RBs on the RB queue are always purged to the level of RB associated with the ESTAE exit prior to the scheduling of the retry routine.

ESTAE retry routines get control in the key in which the ESTAE macro instruction was issued. Retry routines should not reissue ESTAE to maintain the same exit. They may, however, issue ESTAE to add or change exits.

Like the exit routine, the retry routine must be in storage when the exit routine determines that retry is to be attempted. If not already resident within the program, the retry routine may be brought into storage via the LOAD macro instruction by either the user's program or exit routine.

If an SDWA was obtained, the user has a choice of interfaces to his retry address. The user can set (in the SDWA) the registers he wishes to have and request that they be passed to the retry address by coding RETREGS=YES on the SETRP macro instruction. This alternative is most often used when retrying into mainline processing.

If no SDWA was obtained or if RETREGS=NO was specified on SETRP, only parameter registers are passed to the retry address. This alternative is more often used if a special retry routine is to get control.

The parameter registers are as follows:

If no SDWA was obtained:

register	0	12
register	1	address of the user parameter list established via ESTAE or ATTACH with ESTAI.
register	2	pointer to the PIRL if I/O was quiesced and is restorable; otherwise, 0.
register	14	address of supervisor-assisted exit linkage.
register	15	entry point address of retry routine.

If an SDWA was obtained and the exit did not request register update, or freeing of SDWA:

register	0	0
register	1	address of SDWA.
register	2	unpredictable.
register	14	address of supervisor-assisted exit linkage.
register	15	entry point address of retry routine.

If an SDWA was obtained and the exit did not request register update, but did request freeing of SDWA:

register	0	20
register	1	pointer to the user parameter list established via ESTAE or ATTACH with ESTAI.
register	2	pointer to the PIRL if I/O was quiesced and is restorable; otherwise, 0.
register	14	address of supervisor-assisted exit linkage.
register	15	entry point address of retry routine.

Recovery Routine Guidelines

This section is intended to assist in the writing of recovery routines. The actions a recovery routine should take are highly dependent on the function being recovered, therefore, these guidelines are general and intended to serve as suggestions.

The first consideration is that the recovery routine be beneficial. In general, if a function acquires resources which may be requested by another function, or is not known to be related to the task, a recovery routine should be established to free the resources. An example of this type of resource is storage within a subpool which is not task-related (for example, subpool 231). Another case when a recovery routine should be established is when data areas, queues, data sets, etc. which are used by more than one function are manipulated. The recovery routine in this case should maintain integrity in case of failure.

Recovery routines may also be used to:

- intercept errors and perform clean-up processing.
- intercept expected program checks and perform desired action.
- isolate an error to a particular section of processing and continue further processing if possible.
- intercept its own abends and provide tailored dumps.

The second consideration is what type of recovery routine be established. If the function holds a lock, is physically disabled, or is an SRB, an FRR can be used to intercept errors. If the function is running under a task and holds a lock during some portion of its processing, an

ESTAE could be used to catch errors in its processing, an ESTAE could be used to catch errors in its processing if losing the locked status can be tolerated. (For example, if a lock is used only to read a queue to prevent another from changing it.) Also, if the function is running as an enabled, unlocked, SRB, an ESTAE associated with its related task could be used to catch errors in the SRB. If an FRR is not required, but a recovery routine is necessary, an ESTAE routine should be used.

If the function attaches any subtasks, and recovery of the subtask is necessary, the ESTAI/STAI keyword may be specified as an operand of the ATTACH macro instruction.

FRRs

If it is decided that recovery should be via an FRR, the following information should be reviewed:

- Syntax of the SETFRR macro which is documented in Part II of this manual.
- Guidelines for using SETFRR which are documented in this publication under "Establishing Functional Recovery Routines."
- The interface to functional recovery routines which is described earlier in this section.
- Syntax of the SETRP (SET return parameters) macro instruction which is described in **OS/VS2 Supervisor Services and Macro Instructions** with a description in Part II of this publication.
- Interface to FRR requested retry routines which is described earlier in this section.
- The contents of the SDWA (mapped by the IHASDWA mapping macro instruction). All error/recovery information available to an FRR is contained in this work area, and the commentary in this data area serves as the documentation. This data area is described in the **OS/VS2 System Programming Library: Debugging Handbook.**

Task Recovery

If it is decided that recovery via FRR is not necessary, ESTAE recovery should be used. Discussions earlier in this section concerning STAE routines are primarily documented as they are supported for OS/VS2 Release 1 compatibility.

Before designing an ESTAE routine, the following information should be reviewed:

- The syntax of the ESTAE macro instruction which is described in **OS/VS2 Supervisor Services and Macro Instructions** with a description of restricted parameters in Part II of this publication.
- Rules concerning establishing recovery routines (ESTAE/ESTAI exit routines) earlier in this section.
- The interface to ESTAE/ESTAI exits which is documented in this section under "Interface to Functional Recovery Routines".
- Syntax of the SETRP macro instruction which is described in **OS/VS2 Supervisor Services and Macro Instructions** with a description of restricted parameters in Part II of this publication.
- Interface to ESTAE/ESTAI retry routines described earlier in this section.
- The contents of the SDWA (mapped by the IHASDWA mapping macro instruction). All error/recovery information available to an ESTAE/ESTAI exit is contained in this work area and the commentary in this data area serves as the documentation. This data area is described in **OS/VS2 System Programming Library: Debugging Handbook.**

For any recovery routine that is being written, the following information should be reviewed:

- When a task recovery routine receives control it should first examine the code in register 0 to see if a SDWA was provided. If a SDWA was not provided (register 0=12) a save area is not pointed to by register 13 and the registers should not be saved. (Affects task recovery routines--ESTAE/ESTAI.)

- An SDWA is always provided to FRRs. (Affects functional recovery routines.)
- A recovery routine should not assume the registers in the SDWA are its own. Many reasons preclude this such as errors in called routines which have no recovery, errors in an asynchronous routine (for example SRB and IRB). The safest method to assure a successful retry is to save volitile information in the parameter list passed to the recovery routine and use those registers, addresses, and so forth, for retry. (Affects both task recovery routines--ESTAE/ESTAI, and functional recovery routines.)
- If an FRR requests a dump via SETRP, the following should be considered:
  1. No dump will be taken if retry is performed before the error is percolated to task recovery.
  2. The dump will be taken if all FRRs percolate and no subsequent recovery routine suppresses the dump.
  3. The dump will be taken at the task recovery level (after the system is enabled). If volitile information is required, an SDUMP should be issued with the volitile information moved into the 4K SQA buffer described later in this section.

  (These considerations affect functional recovery routines.)
- Dump options are merged during percolation. Those specified on ABEND or SETRP are not used exclusively. (Affects both task recovery routines--ESTAE/ESTAI, and functional recovery routines.)
- The following is additional information about some SDWA fields:
  SDWAPERC - indicates this recovery is being percolated to; however, this does not indicate if a task recovery routine is being percolated to from an FRR.
  SDWAFMID - is zero if recovery is taking place in the address space which suffered the error.
  SDWACLUP - indicates retry is not permitted; resources should be freed in exit.
  (These affect both task recovery routines--ESTAE/ESTAI, and functional recovery routines.)
- An ESTAE exit may request, via a SETRP parameter, that the control program free the SDWA instead of freeing it itself in a retry routine. (Affects task recovery routines--ESTAE/ESTAI.)
- An ESTAE exit may specify, via a SETRP parameter, what the contents of its registers should be on entry to the retry routine.
  (Affects task recovery routines--ESTAE/ESTAI.)
- An ESTAE exit remains in effect when its retry routine receives control. It need not reissue the ESTAE to reestablish itself. (Affects task recovery routines--ESTAE/ESTAI.)

When using ESTAE/ESTAI routines the following should be considered:

- ESTAE or ATTACH may be issued by the routine.
- If an ESTAE/ESTAI exit requests termination or fails ESTAE/ESTAI percolation and the accumulation of dump options occurs.
  - The asynchronous exit indicator will be reset according to the new exit's request.
  - I/O options for the new exit will be ignored.
  - A new SDWA will be initialized.
  - The new exit will be scheduled.
- If all recovery exits (STAE/STAI and ESTAE/ESTAI) fail or indicate termination, the task is terminated.
- If a non-jobstep task issues ABEND with the STEP option, exits are entered for the non-jobstep task. If retry is not requested, the jobstep is terminated with the ABEND code, and only TERM exits will be entered.
- ESTAE exits receive control with the same status (supervisor or problem state) that existed at the time the program issued the ESTAE macro instruction to queue the exit. ESTAE exits created by a program running under any control program protection key

(keys 0-7) receives control in key 0; otherwise ESTAE exits receive control with the same protection key as the program that established the exit.

- ESTAI exits receive control in the key of the TCB of the task that created them.
- In the following cases:
  - forced logoff
  - job step timer expiration
  - wait time limit for job step exceeded
  - ABEND condition because of DETACH of an incomplete subtask
  - ESTAI was issued by a subtask and the attaching task abnormally terminates

  The following actions will occur:

  - ESTAE exit routines will be scheduled if TERM=YES was specified as a parameter when ESTAE was issued.
  - All such routines which may exist will get control in LIFO order.
  - Any ESTAI exit previously suppressed via return code 16, or any exit previously entered which specified return code 0, will not be entered again during TERM processing.
  - Retry indications on return will be ignored.
  - If the TERM option is used on the ESTAE macro instruction issued by an ESTAE exit, it will be ignored.

Although the ESTAE routines should issue SETRP to allow the system to free the SDWA, the freeing could also be accomplished by the retry routine. In this case, it is important to note that the ESTAE recovery routine created under any control program protection key will receive an SDWA in key 0 storage. Therefore, if the retry routine is executing under a key other than key 0, it must issue MODESET to become key 0 before issuing the FREEMAIN.

Clean-Up Routines

Task and address space termination is the process of removing a task or address space from the system, releasing the resources from the task or address space, and making the resources available for reuse. It is the responsibility of the resource managers invoked to establish clean-up routines to 'clean up' the queues and control blocks associated with the resources.

The responsibilities of the clean-up routines include:

- For task termination, removing all traces of the fact that the TCB for the terminating task at one time was connected to, allocated to, or associated with the resource in question. The resource should be left in such a state that it can be reused by another task in the address space or in the system.
- For address space termination, releasing all system queue area and common storage area control blocks obtained for the use of the terminating address space. Also, any buffers, bit settings, pointers, and so on relating to the terminating address space should be reset to make the system appear as if the ASID or ASCB of the terminating address space never existed.

The clean-up routine is also responsible for establishing a recovery environment when first entered to protect itself against errors during its own processing. For SRBs, the clean-up routine issues the PURGEDQ macro instruction to ensure that all undispatched SRBs are removed from the SRB dispatching queue.

Support for Installation-Written Clean-Up Routines

In order to support installation-written clean-up routines, a CSECT is provided into which an installation can assemble the names of subsystem clean-up modules. These modules are given control at the beginning of both the task and address space termination processes to do any special clean-up processing required by the subsystems. (The processing described above is performed by the IBM system routines.) After the CSECT is assembled, it is used to replace the existing CSECT IEAVTRML in load module IGC000IC in SYS1.LPALIB. The installation-written modules must be placed in LINKLIB or a library concatenated to LINKLIB via a LNKLSTxx member of PARMLIB.

Initially, the CSECT consists of four 12-byte entries of all zeros. Each of the first three 12-byte entries is to contain a module name in the first 8 bytes; the last 4 bytes of each entry are reserved and should contain zeroes. The last entry is to consist of all zeroes.

A typical entry for the CSECT may appear as follows:

```
DC   CL8'MODULENM'
DC   XL4'00'
```

Programming Considerations

All clean-up routines of the resource manager use a standard interface, available through the IHARMPL mapping macro. On entry to the clean-up routines, the register contents are as follows:

register	contents
1	pointer to a 4-byte field that contains the address of the interface block
13	pointer to a standard save area
14	return address
15	entry point of clean-up routine
0,2-12	unpredictable

Registers 0-14 must be saved and restored by the clean-up routine; register 15 is used to pass a return code back to termination. A return code of 0 indicates a successful clean-up, and a return code of 4 indicates an unsuccessful clean-up.

The clean-up routines receive control on all task and address space terminations prior to any of the control program resource manager, and receive control in key 0, supervisor state, with no locks held. Each clean-up routine must acquire and release any locks it may need to do its processing.

For task termination, the clean-up routine executes under the TCB of the terminating task, and executes in the address space of the terminating task.

For address space termination, the clean-up routine executes under a task in the address space of the master scheduler. The clean-up routine will be able to examine the ASCB for the address space, queues, and other control blocks which reside in the common area; nothing in the private area for the terminating address space is accessible.

Dumping Virtual Storage

The SNAP and ABEND macro instructions can be used to request dumps, and can be issued by any user. (These macro instructions are described in **OS/VS2 Supervisor Services and Macro Instructions.**) In addition, the system programmer can also use the SDUMP macro instruction to provide dumps of virtual storage, and the CHNGDUMP command to influence the contents of the dump.

Using the SDUMP Macro Instruction

The SDUMP macro instruction can be used by system routines to provide fast unformatted dumps of virtual storage. SDUMP invokes SVC DUMP to provide the services. Only one SVC DUMP may be taken in the system at any one time. Issuers of SVC DUMP with entry by SVC must be authorized via APF or have a control program key.

The SVC DUMP routine can schedule a dump in the address space specified by the ASID parameter of SDUMP. If the user cannot issue an SVC, this service can be initiated by a branch entry, if desired. If the branch entry is used, the branch entry caller must be key 0, supervisor state, and must be in SRB mode, or own a lock, or be disabled (with supervisor bit on). The branch entry interface is by standard linkage conventions. Branch entry callers must issue the CVT mapping macro instruction with the PREFIX=YES parameter.

The SVC DUMP routine with entry by SVC can also schedule a dump from the address space from which it is issued. The caller of this service only must be authorized via APF or have a control program protection key.

SQA Buffer

A 4K buffer is reserved in the system queue area for the callers of SVC DUMP. A user may reserve the buffer and fill it with information before invoking SVC DUMP. The buffer should be used by routines that are involved with volatile data which would be changed or must be changed before SVC DUMP can dump it.

The buffer is pointed to from the CVTSDBF field of the CVT. Since the buffer is for use for all callers of SVC DUMP, it must be treated as a serially reusable resource. The high order bit of CVTSDBF must be checked prior to using the buffer. If the bit is set, it must be assumed that a dump is in progress and the caller must continue processing as if a dump could not be taken.

The first word in the buffer is the actual virtual storage address of the data. The next halfword is the length of the data. A copy of the data follows this 6-byte descriptor field. More 6-byte fields and data may then be specified in this buffer. If the entire buffer is not filled, the last data area must be followed by 6-byte zero descriptor field to indicate the end of meaningful data.

Using the CHNGDUMP Command

The CHNGDUMP operator command is normally not scheduled by the operator, but is scheduled by the system programmer to override the current system dump options for SYSABEND, SYSUDUMP, and SDUMP dumps. The system obtains its normal dump options as follows:

- SDUMP requests -- the options in the system are the ones indicated in the SDUMP parameter list.
- SYSABEND or SYSUDUMP user dump requests -- the system obtains its dump options by merging the dump options in the IEAABD00 or IEADMP00 member of SYS1.PARMLIB with the options indicated on the DUMPOPT parameter of the ABEND, SETRP, and CALLRTM macro instruction.

In order to tailor the dump, the dump option merging may take place in several successive stages. Suppose an error occurred at a low program level, and the recover exit at that level specified certain dump options on a DUMPOPT parameter. Suppose also that there are two other recovery exits between the lowest level exit and the recovery exit that actually precipitates the dump. One of these two exits also specifies certain dump options on a DUMPOPT parameter.

As the recovery effort percolates up toward the top level recovery exit involved, all the dump options from all the exits specifying them merge with the parmlib options to produce the combination of options for the dump the system actually takes.

The CHNGDUMP command can override any or all of these options, regardless of their level of origin. There are two major parameters for this command, as follows:

- SET -- to override the existing dump options.
- DEL -- to delete any or all overriding dump options set by a previous CHNGDUMP SET command.

The overriding options indicated with the SET parameter are the only ones that the system will use for all subsequent jobs and tasks for the life of the IPL, or until they are nullified with the DEL parameter in a subsequent CHNGDUMP command.

Virtual Storage Management

The GETMAIN and FREEMAIN macro instructions allocate and free one or more areas of virtual storage. Although most of the functions of these macro instructions are available to any user, several functions of the GETMAIN and FREEMAIN macro instructions are available only to programs executing in supervisor state under protection key zero.

One of the functions permits a branch entry to the GETMAIN and FREEMAIN routines, rather than an SVC entry. Although use of this function requires additional work on the part of the user, the branch entry is significantly more efficient than the SVC entry and does save some system overhead. This function is provided via the BRANCH parameter.

Another function available only with the branch entry function, allows the user, executing in key zero, to specify the actual key in which the requested storage is to be obtained. This function is provided via the KEY parameter.

The BRANCH Parameter

Branch entry is accomplished by specifying BRANCH=YES on the GETMAIN or FREEMAIN macro instruction. If the BRANCH parameter is used, the caller must preload register 4 with the TCB address, preload register 7 with the ASCB address, and hold the local lock prior to entry. (***Note:*** If the BRANCH parameter is not used, it is still necessary for the current branch entry user of the macro instruction to alter his code to include the preloading of the ASCB address in register 7, and to hold the local lock.)

An additional branch entry point (GLBRANCH) is provided to obtain global storage without the need for holding the local lock. This entry point is available to programs that contain no references to particular address spaces (for example, timer routines). It is necessary, however, to hold the SALLOC lock before entering the routine. Although the TCB address and ASCB address are not required for this entry, register 4 must be loaded with the address of the global save area pointed to by the CVT; this will be done by the macro expansion.

GLBRANCH may be obtained by coding BRANCH=(YES,GLOBAL) on the GETMAIN or FREEMAIN macro instruction that includes the positional parameter RC or RU. The subpools that are supported by this entry are limited to the global subpools - common service area subpools 227, 228, 231 and 241, and system queue area subpools 239 and 245. Any other subpool will be considered an error.

The KEY Parameter

Since branch entry users are required to be executing in key zero at entry time, the KEY parameter satisfies the need to specify the actual key in which the requested storage is to be obtained.

The KEY parameter applies only to six new subpools - 227, 228, 229, 230, 231, and 241. These subpools allow both global and local storage to be obtained in the requestor's storage protection key. Subpools 227 (fetch protected) and 228 (not fetch protected) are fixed global storage in the common service area, and must be freed explicitly. Subpools 229 (fetch protected) and 230 (not fetch protected) are local storage allocated from the top of the private area downward and intermixed with LSQA and SWA, and are freed automatically when the task terminates. Subpools 231 (fetch protected) and 241 (not fetch protected) are global storage in the common service area, and must be freed explicitly.

Real Storage Management

The real storage manager (RSM) administers the use of real storage and directs the movement of virtual pages between auxiliary storage and real storage in page size (4096 bytes) blocks. It makes all addressable virtual storage in each address space appear as real storage. Only virtual pages necessary for program execution are kept in real storage; the remainder reside on auxiliary storage. RSM employs the auxiliary storage manager (ASM) of the data manager to perform the actual paging I/O necessary to transfer pages in and out of real storage. ASM also provides DASD allocation and management for paging I/O space on auxiliary storage. RSM relies on the system resource manager (SRM) for guidance in the performance of some of its operations.

RSM assigns real storage page frames upon request from a pool of available frames, thereby associating virtual addresses with real storage addresses. Frames are repossessed upon termination of use, when freed by a user, when a user is swapped-out or when needed to replenish the available pool. While a virtual page occupies a real storage frame, the page is considered pageable unless specified otherwise as a system page that must be resident in real storage. RSM also allocates virtual equals real (V=R) regions upon request by those programs that cannot tolerate dynamic relocation. Such a region is allocated contiguously from a predefined area of real storage and is non pageable.

The paging services provided include the following:

- PGFIX -- Fix virtual storage contents.
- PGFREE -- Free virtual storage contents.
- PGLOAD -- Load virtual storage areas into real storage.
- PGOUT -- Page out virtual storage areas from real storage.
- PGRLSE -- Release virtual storage contents.

The PGFIX and PGFREE functions are available only to authorized system functions and users. The PGLOAD, PGOUT, and PGRLSE are not restricted and are available to all users; these functions are described in **OS/VS2 Supervisor Services and Macro Instructions.**

Fixing/Freeing Virtual Storage Contents

The PGFIX and PGFREE macro instructions provide complementary functions. The PGFIX macro instruction makes specified storage areas resident in real storage and ineligible for page-out as long as the virtual address space of the requesting TCB remains in real storage. The PGFREE macro instruction makes specified storage areas, which were previously fixed via the PGFIX macro instruction, eligible for page-out. Real frames fixed by PGFIX are not considered pageable until the same number of PGFREE and PGFIX requests have been issued for any virtual area.

Page fixing ties up valuable real storage and is usually detrimental to system performance unless utilization of the resources is extremely high.

In the PGFIX function, you have the option of specifying the relative real time duration anticipated for the fix. If you specify LONG=Y, the duration of the fix will be relatively long. (As a rule of thumb, the duration of a fix is considered long if the interval can be measured on an ordinary timepiece-that is, in seconds.) Additional processing may be required to avoid an assignment of a frame to the V=R area or an area that might be varied offline. If you specify LONG=N, the time duration of the fix is assumed to be relatively short. A long-term PGFIX is assumed if this option is not specified.

In both the PGFIX and PGFREE functions, you have the option of specifying that the contents of the virtual area are to remain intact or be released. If the contents are to be released, you specify RELEASE=Y; otherwise, you specify RELEASE=N. If you specify PGFIX with RELEASE=Y, the PGRLSE function will be performed before the PGFIX function.

If you specify PGFREE with RELEASE=Y, the PGFREE function will be performed and those pages of the virtual subarea with zero fix counts will be released; that is, the contents of virtual areas spanning entire virtual pages that were fixed are expendable and no page-outs for these pages are necessary.

Note: PGFIX does not prevent pages from being paged out when an entire virtual address space is swapped out of real storage. Consequently, the user of PGFIX cannot assume a constant real address mapping for fixed virtual areas in most cases.

Completion Considerations

Under normal circumstances, you can reverse the effect of a PGFIX via a PGFREE when the need for a PGFIX ceases. However, a PGFIX request will sometimes complete asynchronously if it requires a page-in operation. In such cases, it may be necessary to explicitly purge PGFIX operations.

For this reason, the PGFIX function provides a mechanism for signalling event completion. The mechanism is the standard ECB together with WAIT/POST logic. The requestor supplies an ECB address and waits on the ECB after a request. The ECB is posted when all requested pages are fixed in real storage.

Explicit purging of a PGFIX is carried out in one of two ways:

- If the PGFIX is known to be complete, the PGFIX is reversed through the usual PGFREE function.
- If there is any possibility that the PGFIX has not been posted as complete, the PGFREE should be issued with an ECB address supplied. This ECB parameter identifies the event control block that was supplied as an input parameter with the PGFIX being purged. Note that for the purpose of canceling a PGFIX request that has not yet completed, the ECB must uniquely identify the PGFIX request. Consequently, to provide for explicit purging, you must ensure that the ECB address for any incomplete PGFIX can be located in a purge situation, and that the ECB has not been reused at the time the PGFIX is to be canceled.

The PGFREE function always completes immediately and requires no ECB address except for purging considerations.

Virtual Subarea List (VSL)

The virtual subarea list provides the basic input to the page service functions: PGFIX, PGFREE, PGLOAD, PGRLSE, and PGOUT. The list consists of one or more doubleword entries, each entry describing an area of virtual storage. The list must be nonpageable (for example, in SQA or LSQA) and contained in the address space to be processed.

Each parameter list entry has the format shown in Figure 12.

```
Byte    0       1          2   3   4       5   6   7
        FLAGS   START ADDRESS      FLAGS   END ADDRESS + 1
```

Figure 12. Virtual Subarea List Entries

Byte 0 Flags:

Bit 0	(1... ....)	This bit indicates that bytes 1-3 are a chain pointer to the next VSL entry to be processed; bytes 4-7 are ignored, but the checking of this bit is subject to the setting of byte 4, bit 1. This feature allows several parameter lists to be chained as a single logical parameter list.
Bit 1	(.1.. ....)	PGFIX is to be performed; reserved, set by macro instruction.
Bit 2	(..1. ....)	PGFREE is to be performed; reserved, set by macro instruction.
Bit 3	(...1 ....)	PGLOAD is to be performed; reserved, set by macro instruction.
Bit 4	(.... 1...)	PGRLSE is to be performed; reserved, set by macro instruction.
Bit 5	(.... .1..)	Reserved.
Bit 6	(.... ..1.)	Long-term PGFIX is to be performed; reserved, set by macro instruction.
Bit 7	(.... ...1)	Reserved.

Start Address:
The virtual address of the origin of the virtual area to be processed.

Byte 4 Flags:

Bit 0	(1... ....)	This flag indicates the last entry of the list. It is set in the last doubleword entry in the list.
Bit 1	(.1.. ....)	When this flag is set, the entry in which it is set is ignored. This bit takes precedence over byte 0, bit 0.
Bit 2	(..1. ....)	Reserved.
Bit 3	(...1 ....)	This flag indicates that a return code of 4 was issued from a page service function other than PGRLSE.
Bit 4	(.... 1...)	Reserved.
Bit 5	(.... .1..)	PGOUT is to be performed; reserved, set by macro instruction.
Bit 6	(.... ..1.)	KEEPREAL option of PGOUT is to be performed; reserved, set by macro instruction.
Bit 7	(.... ...1)	Reserved.

End Address + 1:
The virtual address of the byte immediately following the end of the virtual area.

Reconfiguration Using Vary Storage

Vary storage permits users of multiprocessing IBM System/370 Models 158 and 168 to more easily specify reconfigurable storage units for use with the other CPU as a uniprocessor. To identify the storage units that the operating system will attempt to preserve for reconfiguration, define the reconfigurable storage unit parameter (RSU=nn) in the IEASYSxx member, where nn is the number of storage units you would like to be able to take offline for the reconfiguration.

Note: If power may be turned off for one CPU, all of the high address storage units, except the highest unit, must be assigned to that CPU. For example, in a system with eight storage units (0-7), units 3-6 should be attached to the CPU that may have power turned off.

The system will attempt to preserve the frames in the storage units defined by the RSU=nn parameter for short-term pages and will be known as the non-preferred area. The amount of time a short-term page is assigned to a processor storage frame should not be measurable on an ordinary time-piece. In other words, the frames of processor storage will be in use for a short period of time like a few hundredths of a second. Frames used for long-term resident storage are clustered in a preferred area. These long-term pages include SQA, non-swappable LSQA, and non-swappable long-term page data.

Figure 13 indicates that when the system is IPLed, the nucleus and V=R areas are assigned to the low-end of processor storage and the SQA is assigned to the high-end of processor storage. The operating system then defines the low and high end processor storage units as preferred areas (in this case 0, 1, and 7). Next, the non-preferred area is defined as requested by the RSU=nn specification and coinciding with the next available high-end processor storage unit (6). Offline processor storage units (5 and 2) are spanned at this time, but omitted. If the RSU=nn parameter value exceeds the available processor storage units, then all units are defined as non-preferred except those previously assigned as preferred at IPL for the nucleus, V=R and SQA.

During IPL, and after the non-preferred area has been defined, the operating system defines the remainder of the processor storage units as the preferred area, for long-term resident frames. In figure 13, RSU=2 was specified, making processor storage units 6 and 4 non-preferred. Any processor storage units varied online after IPL are designated as non-preferred. The remaining processor storage unit (3) is defined preferred.

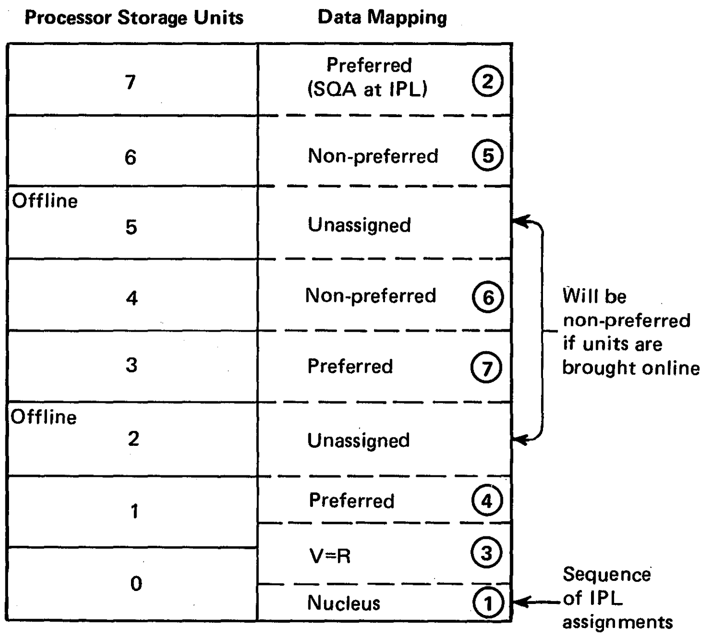


Figure 13. IPL Designation of Processor Storage Units

While jobs are processing, short-term pages are assigned to any available processor storage frame in either non-preferred or preferred areas. Long-term pages are assigned only to processor storage frames in the preferred area. A condition may arise where a long-term page requires processor storage space and there are no preferred area frames available. If this situation occurs, one of the following will result:

- If a short-term page in the common area, or the private area of the current available storage that has not been changed is using a frame in preferred storage, that short-term page is removed and the new long-term page is assigned to the vacated space. This is called immediate steal.
- If all of the frames in the preferred area are being used for long-term pages, the operating system looks for a frame in the non-preferred area. Finding such a frame, the entire processor storage unit is converted from a non-preferred area to a preferred area. This is called dynamic expansion.

Whenever a non-preferred area is converted into a preferred area, the system operator cannot expect to place as many processor storage units offline as were originally designated by the RSU=nn parameter. Therefore, the first time this conversion takes place, message IEA988I is issued to the system console notifying the operator that the preferred area has expanded and that reconfigurability may be impaired. The operator can then determine the processor storage units that are still being preserved for reconfiguration by using the DISPLAY MATRIX (D M) command.

Multiprocessing Configuration Considerations

On a 168 MP, and 158 MP Model 3 the CPUs and the storage (in two but not necessarily equal portions) are each separately powered. On the 158 MP Model 1, however, each CPU and its half of the storage are powered by the same source. If a CPU is powered down, its half of storage is also powered down. Therefore, in a 158 MP Model 1 system, if an installation intends to vary one CPU (CPU 0) offline for maintenance, the user should assign both high and low address ranges to the other CPU (CPU 1). This allows CPU 0 to be powered down without requiring an IPL.

Notes:

- The highest (perferred storage area) and lowest address ranges cannot be varied offline.
- When assigning high and low address ranges to the powered up CPU, ensure (1) that the highest address range is large enough to include all the SQA and preferred storage area, and (2) that the lowest range is large enough to contain all of the nucleus.
- Since the 158 MP Model 3 uses the Alternate Power Down feature to supply power independently to both CPUs, and storage, its associated console and channels can be varied offline and powered on and off without an IPL. Its storage can remain online to the powered up CPU.

If a 158 MP or 168 MP is configured as two MVS uniprocessors, both systems should be configured at the highest and lowest addresses for an MP system, to allow reconfiguration to an MP system without IPL. This will be done automatically by specifying STORAGE=highest address in the CTRLPROG macro instruction during system generation. For example, in a 4-megabyte system, each uniprocessor should be configured at the following addresses:

0-512
512-1024
3072-3584
3584-4096

To accomplish this, you would specify STORAGE=4,096,000. This leaves a 2-megabyte "hole" for reconfigured storage; to reconfigure to an MP system, you can fill this hole with the following address ranges by first adjusting the hardware configuration panel and then entering the proper VARY commands:

1024-1536
1536-2048
2048-2560
2560-3072

Note: The 158 MP Model 1 and 158 MP Model 3 are compatible and can be interconnected in an MP configuration. However, when these models are interconnected the Alternate Power Down and Asymmetric storage features on the 158 MP Model 3 are not functional.

If there is more than one console, one must be specified as the primary console. One reason is that the communications task must know which to use during IPL. Do not use the same address for both consoles. In general, no two devices should have the same address. If identical addresses are used, failure of one console is considered failure of both; with 3066 consoles, errors on one can result in alteration of the display on the other.

In a multiprocessing environment, the concurrent execution of tasks makes it difficult to predict the contents of dynamic fields in all system control blocks, (that is, fields whose contents change during processing from IPL time). Thus, all read-only and write-only references in user programs to these dynamic fields must be evaluated for their impact on the user's program and/or the system control program, and should be changed or eliminated accordingly. Otherwise, the effect (such as program abend or system failure) is dependent upon the user program's recovery scheme and/or system control program's use of that changed field.

Miscellaneous Services

Additional services are provided by the supervisor which do not appropriately fit into the previous chapters. These services, discussed in this chapter, include:

- operator messages (WTO and WTOR macro instructions)
- user-written message routing exit routines
- service management facilities (SCHEDULE and PURGEDQ macro instructions)
- stage 1 exit effector (CIRB macro instruction)
- user-written SVC routines
- missing interruption handler
- power warning feature support

Writing Operator Messages

The WTO and WTOR macro instructions allow you to write messages to the operator. The WTOR macro instruction also allows you to request a reply from the operator. A complete description of the use of these two macro instructions is found in **OS/VS2 Supervisor Services and Macro Instructions.**

Routing the Message

The WTO and WTOR macro instructions have two special parameters, MSGTYP and MCSFLAG. The MSGTYP parameter specifies how the message is to be routed; the MCSFLAG parameter specifies that the macro expansion is to set bits in the MCSFLAG field as indicated by each name coded. Only programmers familiar with MCS should use these parameters, since using them inproperly could impede the entire message routing scheme.

If MSGTYP=Y is specified, the message type specifies that two bytes are to be reserved in the WTO or WTOR macro expansion so that flags can be set to describe what MSGTYP functions are desired. If Y is specified, two bytes of zeros are to be included in the macro expansion at displacement WTO (or WTOR + 8) + 12 + the total length of the message text, descriptor code, and routing code fields. If MSGTYP=N is specified, or if the MSGTYP parameter is omitted, the two bytes are not needed and the message will be routed as specified in the ROUTCDE parameter.

The bit definitions for MSGTYP=Y are:

Bit 0	MONITOR JOBNAMES
Bit 1	MONITOR STATUS
Bit 2-4	Reserved
Bit 5	MONITOR SESS
Bit 6-15	Reserved

When MSGTYP=Y is specified, the issuer of the WTO or WTOR macro instruction that contains the MSGTYP information must set the appropriate message identifier bit in the MSGTYP field of the macro expansion. The MCSFLAG field in the macro expansion has been set to zero, indicating that the MSGTYP field is to be used for the message routing criteria. When the message type is identified by the system, the message will be routed to all consoles and TSO terminals in operator modes that had requested that particular type of information. If there are no consoles or terminals requesting that particular type of information, the WTO message will not be sent anywhere; however, a WTOR message will be sent to the master console. The routing codes and REG0 MCSFLAG field, if present, will be ignored.

Writing a Multiple-Line Message

The WTO macro instruction is used to write a multiple-line message to one or more operator consoles. System programs (supervisor state, protection key 0-7, or APF-authorized) may create a multiple-line message with more than one WTO request.

The first WTO request supplies the first lines of the message. Other WTO requests can then add lines to the message. The additional lines would appear at the end of the message, and would continue until an 'END' line is added.

For the first request in a multiple-line sequence, the leftmost three bytes of register zero must by zero; you must ensure that this is done.

The first request receives a message identifier back in register 1. To add more lines, the next multiple-line request must have this identifier in the leftmost three bytes of register zero.

If the conditions on register zero are not met, it would appear to SVC 35 (WTO) that the multiple-line request is adding lines to an existing message, and the new message will not be created.

Message Routing Exit Routines

This topic provides detailed information on how to write user exit routines that modify the routing and descriptor codes of WTO or WTOR messages for the VS2 operating system. Information is provided on inserting this exit routine into the resident portion of the control program. In addition, a description of the characteristics and configuration of MCS is supplied.

Characteristics of MCS

The multiple console support (MCS) facility routes messages to different functional areas according to the type of information that the message contains. In MCS, a functional area is defined as one or more operator's consoles that are doing the same type of work. (Some examples of functional areas are: (1) the tape pool area, (2) the disk pool area, and (3) the unit record pool area.) Each WTO and WTOR macro instruction is assigned one or more routing codes which are used to determine the destination of the message. There are fifteen routing codes that can be used. When the message is ready to be routed, the routing codes assigned to the message are compared to the routing codes assigned to each console. If any of the routing codes match, the message is sent to that console.

If the standard routing codes provided on application and system messages do not cover special situations at an installation, the routing codes can be modified by coding a user exit routine. The exit routine receives control prior to the routing of messages so users can examine the message text and modify the message's routing and descriptor codes. The system will use the modified routing codes to route the message. Descriptor codes provide a mechanism for message presentation and deletion, and are explained later in this chapter.

Automatic console switching occurs when permanent hardware errors are detected. Command-initiated console switching is provided to permit restructuring of the system console configuration and the hard copy log by system operators. Consoles can be moved into or out of functional areas at any time during system operation.

A hard copy log records messages, operator and system commands, and operator and system responses to commands. The hard copy log can be a console device or it can be the system log (SYSLOG). The number and type of messages recorded on the log is optional. The installation may wish to record a selected group of messages, or it may wish to record all messages. If commands are recorded, the system automatically records command responses.

Whenever possible, the hardcopy function should be delegated to an output-only device (such as a printer) or to the system log.

Programming Conventions for WTO/WTOR Routines

The programming conventions for the WTO/WTOR exit routines are summarized below:

- Exit routine is part of the resident control program. The program should be loaded on a page boundary.
- Exit routine is any size.
- Exit routine may allow interruptions. The routine will receive control with no locks held; it should return control with no locks held.
- Exit routine is reenterable and serially reusable. Macro instructions whose expansions store information into an online parameter list should not be used.
- IEECVXIT is name of routine.
- Registers must be saved at entry and restored prior to returning.
- Exit routine may issue WAIT, XCTL, WTO, or WTOR macro instructions.

  Note: The ***WAIT*** macro instruction should not be used when the exit routine is entered under the console communications task. Doing so will permanently terminate console communications.
- Exit routine is part of WTO SVC. If the exit routine terminates abnormally, the WTO request will be terminated.
- Exit from the routine is via the RETURN macro instruction.
- Format of text and codes is:

  Message text (128 characters padded with blanks).

  Routing codes (4 bytes). Descriptor codes (4 bytes).

In the routing code field, a bit setting of "1" indicates that the WTO or WTOR was assigned that particular routing code. Bit assignments and their meanings are:

Bit	**Assignment**	**Meaning**
Byte 0		
Bit 0	Routing code 1	Master Console Action
Bit 1	Routing code 2	Master Console Information
Bit 2	Routing code 3	Tape Pool
Bit 3	Routing code 4	Direct Access Pool
Bit 4	Routing code 5	Tape Library
Bit 5	Routing code 6	Disk Library
Bit 6	Routing code 7	Unit Record Pool
Bit 7	Routing code 8	Teleprocessing Control
Byte 1		
Bit 0	Routing code 9	System Security
Bit 1	Routing code 10	System Error/Maintenance
Bit 2	Routing code 11	Programmer Information
Bit 3	Routing code 12	Emulators
Bit 4	Routing code 13	Available for Customer Usage
Bit 5	Routing code 14	Available for Customer Usage
Bit 6	Routing code 15	Available for Customer Usage
Bit 7	Routing code 16	Reserved
Byte 2		Reserved
Byte 3		Reserved

In the descriptor code field, a bit setting of "1" indicates that the WTO or WTOR was assigned that particular descriptor code. Bit assignments and their meanings are:

Bit	**Assignment**	**Meaning**
Byte 0		
Bit 0	Descriptor code 1	System Failure
Bit 1	Descriptor code 2	Immediate Action Required
Bit 2	Descriptor code 3	Eventual Action Required
Bit 3	Descriptor code 4	System Status
Bit 4	Descriptor code 5	Immediate Command Response
Bit 5	Descriptor code 6	Job Status
Bit 6	Descriptor code 7	Application Program/Processor
Bit 7	Descriptor code 8	Out-of-Line Message
Byte 1		
Bit 0	Descriptor code 9	DISPLAY or TRACK command response
Bit 1	Descriptor code 10	Dynamic Status Displays
	11 through 16	Reserved
Byte 2		Reserved
Byte 3		Reserved

Messages Not Using Routing Codes

There are certain messages that the exit routine does not see. These are messages that have the MSGTYP parameter in the WTO or WTOR macro instruction coded with the JOBNAMES, STATUS, or Y parameter, multiple-line WTOs (including status displays) and messages that are being returned to the requesting console, for example, a response to a DISPLAY A command. Routing of these messages is on criteria other than the routing codes; therefore, the system bypasses the exit routine.

Writing a WTO/WTOR Exit Routine

To modify the standard routing codes and descriptor codes, a WTO/WTOR Exit Routine must be written. This routine will be part of the control program. If a message's routing code field is used by the operating system to route the message, the routine will receive control prior to the routing of the message. When the routine receives control, register 1 contains a pointer to a word that points to the first word of the message text. The message text field is 128 bytes long, followed by a four-byte routing code field and a four-byte descriptor code field. The exit routine may examine but not modify the message text.

A message will be sent to only those locations specified in the modified routing codes. All messages with modified routing codes are sent to the hard copy log when the log is included in the operating system. When the log is not included, the exit routine must not suppress messages that contain a routing code of 1, 2, 3, 4, 7, 8, or 10 since messages with these codes are necessary for system maintenance. Message suppression is turning off all routing codes of a message by setting the routing code field to zero, thus causing the message to be discarded. WTO messages can be suppressed. If a WTOR message is suppressed, it will be sent to the master console by the operating system.

Adding a WTO/WTOR Exit Routine to the Control Program

The WTO/WTOR exit routine is standard. If the user does not specify one, the IBM-supplied module IEECVXIT is included.

To enter the exit routine into the control program before system generation, the linkage editor should be used to replace the dummy WTO/WTOR exit routine IEECVXIT in SYS1.AOSC5 with the WTO/WTOR exit routine. The linkage editor should be instructed to load IEECVXIT on a page boundary.

To enter the exit routine into the control program after system generation, the linkage editor should be used to replace the dummy WTO/WTOR exit routine IEECVXIT in the SYS1.LPALIB with the user-written WTO/WTOR exit routine.

Service Management

Service Management facilities provide a basic set of system services which allow internal system components to structure themselves to run enabled, nonserialized, and in parallel on a multiprocessing system with considerably less overhead than would be required by using existing task management services. The facilities provided by service management are those services required to:

- Introduce a service request to execute a service routine into the queue of work within the system.
- Perform priority dispatching of the requested service routine.
- Support the needs of recovery/termination for cleaning up the asynchronous processes.

The first two facilities are accomplished via the SCHEDULE macro instruction; the third facility is accomplished via the PURGEDQ macro instruction.

The main features of this support are:

- A control block, called a service request block (SRB), which represents a service request. This block, like a TCB, identifies a unit of work to the dispatcher. The block is smaller than a TCB, and requires less information to be specified about each request.
- The SCHEDULE macro service, which enters SRBs into the dispatchable work queue with a minimum of overhead.
- Changes to the dispatcher to operate under this service request control structure in addition to the old task structure. Maximum performance is provided when dispatching service requests. In addition, SRBs may be scheduled to different address spaces either at a priority equal to the specified address space or at an independent priority higher than any address space.

Service management provides a system-wide dispatching facility which can be used to increase parallel processing on multiple CPUs. The programs benefiting the most from these facilities are those which have independently dispatchable units of work, but are forced to run as a single task.

In addition, service management facilities make the system aware of the smaller units of work and allow the service requests to be dispatched in parallel on multiple CPUs at a higher priority than the task work.

The service management facilities also provide a mechanism which is used for almost all communications between address spaces, and is used to run some parts of interruption handlers as service requests allowing more enablement and parallelism for these services. For example, when an interruption occurs, the interruption handler collects the necessary information about the I/O interruption and schedules a service request block (SRB). The interruption handler can then start I/O requests which were waiting for the I/O path and accept any additional pending interruptions. By delaying complete processing of the interruption, this approach allows faster reuse of channels and lower disabled interruption time.

The scheduling of the SRB provides the ability:

- to complete the interruption process on any CPU, not just the one which took the interruption
- to process the interruption enabled except where specific serialization through locks is used

- to switch from the random address space where the interruption was taken to the address space of the user which originally requested the I/O. This latter capability provides the interruption handler routine with addressability to the user's control blocks necessary to complete the interruption processing.

Scheduling Service Requests

The introduction of a service request into the queue of work is accomplished via the SCHEDULE macro instruction. To use this macro, you must provide the following information:

- The address of a previously obtained and formatted service request block (SRB) that is to represent the request until it is actually dispatched. The contents of the SRB supplied define the attributes of the routine to be given control. Once the service routine is given control, the SRB is no longer needed by the system and may be released.
- The priority of the request relative to other requests in the system. The service may be scheduled at either local or global priorities.

The SCHEDULE macro instruction does not obtain storage for the SRB. It simply causes the indicated SRB to be queued at the appropriate priority. When the request subsequently becomes the highest priority work in the system, the dispatcher will dispatch it in the address space specified by the SRB.

Service Request Blocks (SRBs)

Service requests are represented by service request blocks (SRBs). SRBs must reside in fixed, commonly-addressable storage. These control blocks are supplied by the function requesting a service. The basic information contained in an SRB is documented in the **OS/VS2 System Programming Library: Debugging Handbook,** section 6, "Data Areas". The information required within the SRB at the time of SCHEDULE is as follows:

SRBASCB - contains the address of the ASCB indicating the address space in which the asynchronous routine will be dispatched.

SRBPKF - indicates, in the high-order 4 bits, the protect key that the routine will assume. The low-order 4 bits must be zero.

SRBEP - specifies the address of the entry point of the asynchronous routine.

SRBSAVE - contains the address of a status save area. This field must contain all zeroes.

SRBPARM - contains a fullword to be loaded into register 1 when the asynchronous routine is dispatched.

SRBCPAFF - defines the CPU affinity. If all zeroes, or all ones, no affinity is implied. Otherwise, this field contains a bit mask in which the bits that are set 'on' to indicate on which CPUs the SRB may be dispatched. (The nth bit set 'on' indicates that the SRB may be dispatched on the CPU with physical address n.)

SRBPRIOR - specifies a code representing the priority level to which the SRB is being scheduled. The codes are assigned as follows:

0	SYSTEM
4	NONQ

SRBRMTR - contains the address of a Resource Manager Termination routine. This routine will be responsible for cleaning up SRBs which have been scheduled but not yet dispatched. The address also serves as a function identifier to PURGEDQ when purging SRBs from the queues. If it is possible for the SRB to be purged, either directly by the resource manager, or indirectly by task or memory termination, then this field must contain a valid non-zero address of a routine to clean up that SRB. The routine must be commonly-addressable to all address spaces.

SRBPTCB - contains the address of a TCB associated with the asynchronous routine which serves two purposes:

- If the asynchronous routine encounters a double failure (error in routine and in recovery routine), this task will be scheduled for abnormal termination.
- An identifier to PURGEDQ during the purge process. If this SRB is to be purged during the termination of that task, the address of the associated TCB must be specified in this field. If this SRB is not related to any task, or purging is not necessary, a zero value should be specified.

SRBASID - contains the ASID of an address space associated with the asynchronous routine. If a non-zero value was specified, in the SRBPTCB field, this value must be specified and must contain the ASID of the address space containing that TCB. Otherwise, a zero value may be specified.

Priorities

Services may be scheduled for execution at either global or local priorities. Service requests queued at the global level are given a priority which is above that of any address space, regardless of the actual address space in which they will be dispatched. Service requests at the local level are given a priority equal to that of the address space in which they will be dispatched, but higher than that of any task within that address space.

Within the global and local priorities, there are two additional priority levels. One of the levels is for general system usage; the other (called the nonquiesceable level) is for specialized functions necessary to perform a quiesce of SRBs.

Service requests at the nonquiesceable level continue to be dispatched while the address space is in the process of being quiesced. Requests queued at the system level in an address space will not be dispatched while the quiesced status is in effect. Usage of the nonquiesceable level is restricted in the following manner: At times, it is necessary to stop the dispatching of SRBs in an address space -- that is, prevent new SRBs from being dispatched and allow all SRBs that have already been dispatched to complete their processing. However, some of the dispatched SRBs may have been suspended due to lock requests or page faults. Since page fault processing and rescheduling of the suspended SRBs makes use of SRBs themselves, it is necessary to have a nonquiesceable level at which these SRBs can be scheduled while the other SRBs have been stopped.

Characteristics of Service Requests

Service routines have the following characteristics:

- All routines are entered in supervisor state, enabled, and unlocked. On entry, the address of the SRB is in register 0 and the return address is in register 14. The entry point address is in register 15 and register 1 contains the user field. The routines cannot enter problem program mode and must establish a recovery environment.
- The routines may not issue SVCs. (However, ABEND may be issued.)
- The routines are nonpreemptible -- that is, although the routines are run enabled and may be interrupted by an asynchronous interruption, they will not lose control to higher priority tasks or SRBs until control is given up voluntarily. However, service routines may lose control due to synchronous events which cause suspension of the program in control -- for example, page faults and unconditional requests for suspend-type locks. In this case, full status of the process is saved and other work is dispatched; the service request will be redispatched when the problem is resolved.

- The routines may take page faults. Page faults encountered in the unlocked state will be handled per service request; page faults encountered in the locked state will prevent other processing which requires the lock from proceeding until the page fault is resolved.
- The routines may request any lock through the SETLOCK interface.
- The routines must return control to the address supplied in register 14, and must return control in supervisor state with no locks held. All cleanup must be performed prior to exiting.

Purging Service Requests

When a task or address space terminates abnormally, outstanding requests for the task or address space must also be terminated. The PURGEDQ macro instruction establishes a standard mechanism for purging these requests. To use this macro, you must provide the following information:

- The address space identifier of the address space in which the SRB is scheduled to be dispatched. If none is specified, the current address space is assumed.
- The address space of the task control block of the task associated with the SRB for which the purge is to be performed. If none is specified, the current TCB is assumed in the current address space.
- The address of the resource manager termination routine. If no address is supplied, the purge is performed for all resource managers.

The PURGEDQ routine dequeues all nondispatched SRBs and waits for completion of any active SRBs if the PURGEDQ was for the current address space. After all of the SRBs have been dequeued or completed, the resource manager termination routine specified in the SRB is given control and the required cleanup is performed for each dequeued SRB. No locks should be held when PURGEDQ is invoked.

PURGEDQ Parameters

The inputs to the PURGEDQ macro instruction are specified by the ASID, ASIDTCB, and RMTR parameters.

The ASID parameter specifies the address of a half word containing an address space identifier. PURGEDQ will search for SRBs scheduled to be dispatched into the address space specified by this parameter. If an address space other than the current address space is indicated, only SRBs which have not yet been dispatched will be affected as PURGEDQ will not wait for SRBs already dispatched but not completed. If this parameter is omitted, the current address space will be assumed.

The caller of PURGEDQ can purge the SRBs associated with a specific task by coding the ASIDTCB parameter. The ASIDTCB parameter specifies the address of a doubleword associated with the TCB for which SRBs are to be purged. If the parameter is omitted, the purge will occur for SRBs associated with the current task in the current address space. The following table describes the acceptable values for the ASIDTCB parameter, and the meaning of the values:

Bytes 0 - 7 zero	All SRBs are to be purged.
Bytes 0 - 1 reserved Bytes 2 - 3 nonzero ASID Bytes 4 - 7 zero	All SRBs associated with the specified address space (SRBPASID field) are to be purged.
Bytes 0 - 1 reserved Bytes 2 - 3 nonzero ASID Bytes 4 - 7 nonzero TCB address	All SRBs associated with the specified address space (SRBPASID field) and the specified task (SRBPTCB field) are to be purged.

All other values are unacceptable and will produce unpredictable results.

The RMTR parameter specifies the address of the resource manager termination routine.

The interface to the RMTR is as follows:

Register	Contents
0	Contents of register 0 of the caller of PURGEDQ at the time the PURGEDQ SVC was issued. This provides the ability to pass information from the caller of PURGEDQ to the RMTR routine.
1	SRB address of the dequeued SRB.
2	Contents of SRBPARM of the dequeued SRB.
14	Return address of PURGEDQ.
15	Entry point of RMTR.

The PSW is enabled, in supervisor state with key 0 and no locks held.

The RMTR must not leave supervisor state or issue ESTAE with branch entry. The RMTR must return control enabled, in supervisor state with key 0 and no locks held and via a BR14. It may, however, acquire locks, issue SVCs and destroy input registers.

This routine must be commonly-addressable from all address spaces. It must be unique to the resource manager (that is, it should not be the address of some common system service) if the PURGEDQ issuer desires the routine to receive control only on a unique PURGEDQ. These restrictions are required to insure the uniqueness of the identifier and to allow the RMTR to be invoked from any address space.

Creating Interruption Request Blocks

The CIRB macro instruction causes the Stage 1 Exit Effector routine to create and initialize an interruption request block (IRB). The IRB is used to control an asynchronous user exit routine requested by the caller.

The Stage 1 routine obtains a work area in which the caller may construct interruption queue elements (IQEs), and obtains a register save area in which the user exit routine may later save the registers of the requesting program. The routine obtains space for the IRBs (and IQEs) and the work area from local supervisor queue space. The work area follows and is continguous to the IRB. The register save area, if requested, is obtained from subpool zero of the user program's region, and is therefore not contiguous to either the IRB or its work area.

After the storage for the IRBs and optional work and storage areas is obtained, the Stage 1 Exit Effector routine initializes the IRB. The initialization is accomplished according to the flag bits passed to the routine in register 1.

The information initialized in the IRB includes the save area address, the size of the IRB, the entry point of the user exit, the PSW to be loaded to start execution, and a series of flags communicating actions to be taken when the asynchronous exit routine terminates.

For details on the Stage 1 exit effector, and for information on the Stage 2 and 3 exit effectors, see **OS/VS2 Scheduler and Supervisor Logic.**

Writing SVC Routines

User-written SVC routines become part of the control program, so you must follow the same programming conventions used by SVC routines supplied with VS2. Four types of SVC routines are supplied with VS2, and the programming conventions for each type differ. The general characteristics of the four types are described in the following text, and the programming conventions for all types are shown in tabular form.

Characteristics of SVC Routines

All SVC routines receive control in the supervisor state. You should keep the following characteristics in mind when deciding what type of SVC routine to write:

- Location of the routine -- Your SVC routine can be either in storage at all times as part of the resident control program, or in the fixed or pageable link pack area. Types 1 and 2 SVC routines are part of the resident control program, and types 3 and 4 are in the link pack areas. No transient areas are provided in MVS.
- Size of the routine -- SVC routines are not limited in size, but should be kept under one page if disabled global locks are obtained.
- Design of the routine -- All SVC routines must be reenterable. If you wish to aid system facilities in recovering from machine malfunctions, your SVC routines must be refreshable.
- Authorization -- At nucleus initialization, all SVC routines that are to be loaded into the fixed or pageable LPA must be contained in SYS1.SVCLIB, SYS1.LPALIB, or SYS1.LINKLIB. After the initial load of type 4 SVCs, subsequent loads may be contained in any authorized library.
- Loads -- Type 3 SVCs have one load; type 4 SVCs may have multiple loads. Since type 4 SVCs require XCTL overhead, and since there are no size limitations for type 4 SVCs, type 4 SVCs have no advantage over type 3 SVCs. Unless an SVC is to be used in another system as well as in MVS, type 4 SVC routines should never be used.
- Serialization -- In MVS, locking has generally replaced CPU disablement as the technique for serializing multiple CPU fuctions. If you write SVC routines which must serialize with other parts of the control program, you must use the same locking conventions as the control program. If you write two or more SVC routines which must serialize with each other, you can use either the locking facilities or the ENQ/DEQ services.

SVC routines can receive control with one or more locks held. You must define which locks are to be acquired for your SVC routines during system generation. (For more information on locking, see the discussion under Resource Control.)

SVC routines are normally entered enabled. However, an SVC routine will be entered disabled if it was specified that a disabled spin lock is to be acquired for the routine. (See reference code 2 on the following page.)

Programming Conventions for SVC Routines

The programming conventions for the four types of SVC routines are summarized in Figure 14. Details about many of the conventions are in the reference notes that follow. The notes are referred to by the numbers in the last column of the figure. If a reference note for a convention does not pertain to a specific type of SVC routine, that type is indicated by an asterisk.

Conventions	Type 1	Type 2	Type 3	Type 4	Reference Code
Part of resident control program	Yes	Yes	No	No	
Size of routine	Any	Any	Any	Any	
Reenterable routine	Yes	Yes	Yes	Yes	1
Refreshable routine	No*	No*	Yes	Yes	2
Locking requirements	Yes	No	No	No	3
Entry point	Must be the first byte of the routine or load module, and must be on a doubleword boundary				
Number of routine	Numbers assigned to your SVC routines should be in descending order from 255 through 200				
Name of routine	IGCnnn	IGCnnn	IGC00nnn	IGCssnnn	4
Register contents at entry time	Registers 3, 4, 5, 6, 7, and 14 contain communication pointers; registers 0, 1, 13, and 15 are parameter registers				5
Supervisor request block (SVRB) size	No SVRB exists	200	200	200	7
May issue WAIT macro instruction	No*	Yes	Yes	Yes	8
May issue XCTL macro instruction	No*	No*	No*	Yes	9
May pass control to what other types of SVC routines	None	Any	Any	Any	10
Type of linkage with other SVC routines	Not Applicable	Issue supervisor call (SVC) instruction			11
Exit from SVC routine	Branch using return register 14				12
Method of abnormal termination	ABTERM	ABEND			
Recovery	FRR	ESTAE or FRR			13

Note: Reference code does not apply to items marked with *.

Figure 14. Programming Conventions for SVC Routines

Reference Code	SVC Routine Types	Reference Notes
1	all	If your SVC routine is to be reenterable, you cannot use macro instructions whose expansions store information into an inline parameter list.
2	3,4	Types 3 and 4 in the pageable LPA must be refreshable. Types 3 and 4 in the fixed LPA must be reenterable, but not necessarily refreshable.
3	all	The following conventions on locking requirements apply: • Type 1 SVC routines always receive control with the local lock held and must not release the local lock. Additional locks may be requested prior to entry via the SVCTABLE macro instruction or may be requested dynamically within the SVC routine. • Types 2, 3, and 4 may also request locks via the SVCTABLE macro instruction or may obtain them dynamically. • Types 1 and 2 may request that any locks be held on entry. Types 3 and 4 may only request that the LOCAL or LOCAL and CMS lock be held. • If no locks are held or obtained, or only suspend locks (LOCAL and CMS) are held or obtained, the SVC routine executes in supervisor state, key 0, enabled mode. • If disabled spin locks are held or obtained, the SVC routine executes in supervisor state, key 0, disabled mode. No SVCs may be issued. • SVCs may not take disabled page faults. Therefore, if a disabled spin lock is held, the SVC routines must ensure that any referenced pages are fixed. For types 3 and 4, all pages containing code must be fixed. • An FRR may be defined for any SVC routine that holds or obtains locks to provide for abnormal termination (see reference note 9).
4	all	You must use the following conventions when naming SVC routines: • Types 1 and 2 must be named IGCnnn; nnn is the decimal number of the SVC routine. You must specify this name in an ENTRY, CSECT, or START instruction. • Type 3 must be named IGC00nnn; nnn is the signed decimal number of the SVC routine. • Type 4 must be named IGCssnnn; nnn is the signed decimal number of the SVC routine, and ss is the number of the load module minus one. For example, ss is 01 for the second load module of the routine.
5	all	Before your SVC routine receives control, the contents of all registers are saved. In general, the location of the register save area is unknown to the routine that is called. When your SVC routine receives control, the status of the registers is as follows: • Register 0 and 1 contain the same information as when the SVC routine was called. • Register 2 contains unpredictable information. • Register 3 contains the starting address of the communication vector table (CVT). • Register 4 contains the address of the task control block (TCB) of the task that called the SVC routine. • Register 5 contains the address of the supervisor request block (SVRB), if a type 2, 3, or 4 SVC routine is in control. If a type 1 SVC routine is in control, register 5 contains the address of the last active request block. • Register 6 contains the entry point address. • Register 7 contains the address of the address space control block (ASCB). • Registers 8 through 12 contain unpredictable information. • Register 13 contains the same information as when the SVC routine was called. • Register 14 contains the return address. • Register 15 contains the same information as when the SVC routine was called. You must use register 0, 1, and 15 if you want to pass information to the calling program. The contents of registers 2 through 14 are restored when control is returned to the calling program.

Reference Code	SVC Routine Types	Reference Notes
7	2,3,4	The SVRB is no longer extendable, but is a fixed size of 200 bytes. When a type 2, 3, or 4 SVC routine receives control, register 5 contains the address of the SVRB within this 200-byte area. This SVRB contains a 40-byte "extended save area." In addition, an area is provided for a STAE control block (SCB); this SCB will be used by the recovery termination manager when an ESTAE or ESTAI is issued within an SVC routine.
8	2,3,4	You can issue the WAIT macro instruction if you hold no locks. You can issue WAIT macro instructions that await either single or multiple-events. The event control block (ECB) for single-event waits or the ECB list and ECBS for multiple-event waits must be in virtual stroage.
9	4	When you issue an XCTL macro instruction in a routine under control of a type 4 SVRB, the new load module must be located in the fixed or pageable link pack area. The contents of registers 2 through 13 are unchanged when control is passed to the load module; register 15 contains the entry point of the called load module.
10	all	No SVC routines except ABEND may be called if locks are held. ABEND may be called at any time.
11	all	No locks may be held. If locks are held, branch entry to SVCs is acceptable, or the locks may be freed, the SVC issued, and the locks reobtained.
12	all	Branch using return register 14 should be used. Otherwise, if locks are held, SVC 3 results in abnormal termination.
13	all	If an SVC routine is entered with a lock held or if an SVC routine obtains a lock, it should specify a functional recovery routine (FRR) for a long as the lock is held (see SETFRR macro instruction). The FRR receives control if an error occurs, and ensures the validity of the data being serialized by the lock; the FRR either recovers or releases the lock and continues with termination. If no FRR is specified, the recovery termination manager will release the lock and terminate the task. No cleanup of the data is performed. (Note that the lock is released before any STAI/ESTAI/ESTAE (or STAE) recovery routine is entered. If no locks are acquired for or by an SVC routine, then an ESTAE may be used to define your recovery processing (see ESTAE and SETRP macro instructions).

Inserting SVC Routines Into the Control Program

You insert SVC routines into the control program during the system generation. Before your SVC routine can be inserted the routine must be a member of a cataloged partitioned data set named SYS1.name, where the name is a name of your choice. The data set must be an authorized library (see discussion on APF).

The following text describes the informaion you must supply during system generation. You will find a description of the macro instructions required during system generation in the **OS/VS2 System Programming Library: System Generation Reference** publication.

Specifying SVC Routines

You use the SVCTABLE macro instruction to specify the SVC number, the type of SVC routine, and the locks that are required for your SVC.

Inserting SVC Routines During the System Generation Process

To insert an SVC routine during system generation, the name of the partitioned data set and the names of the members to be included must be specified in the DATASET macro instruction:

- For each type 1 or 2 SVC, a member (containing one or more SVC routines) should be specified in the DATASET macro instruction that is being used to define SYS1.NUCLEUS.

- For each type 3 or 4 SVC, a member (containing only one SVC routine) should be specified in the DATASET macro instruction that is being used to define SYS1.LPALIB.

Type 5 SVC Facility

The Type 5 SVC facility permits a user to indicate (at system generation time) that one or more user-written SVCs will be added to the system after the completion of SYSGEN. This facility is only used in cases where the SVC routines for the user-written SVC are not available at the time of the SYSGEN. This facility further permits the user to specify the authorization characteristic and lock requirements as input parameters to the SYSGEN macro instruction SVCTABLE. The name of the entry point to the SVC routine will be supplied at a later time.

SVC Table Entries

Each entry in the SVC table has the following format:

Offset	Length	Name	Description
0	4	SVCEP	SVC Entry Point Address
4	2	SVCATTP1	Attributes
	....	SVCTP1	Type 1 SVC
	1...	SVCTP2	Type 2 SVC
	11..	SVCTP34	Type 3 or 4 SVC
	.... 1...	SVCAPF	APF Authorized 1-Authorized
	.... .1..	SVCESR	SVC is a part of the ESR
6	2	SVCLOCKS	Lock Attributes
	1...	SVCLL	Local Lock Needed
	.1..	SVCCMS	CMS Lock Needed
	..1.	SVCSRM	SRM Lock Needed
	...1	SVCALLOC	SALLOC Lock Needed
	.... 1...	SVCDISP	Dispatcher Lock Needed

If a Type 5 SVC is specified at system generation time, the following entry will be generated:

IGCRETRN	Entry Point Address (of IGCRETRN)
Type 1	
Authorization	Default is unauthorized, unless the user specifies authorized.
Locks	Default is no locks; user may specify any lock or combination of locks needed by his routine.

Note: IGCRETRN is an existing routine which zeroes register 15 and returns via register 14.

Subsequent to SYSGEN, the user is expected to supply the SVC routine and to update the entry in the SVC Table. Assuming that the other information in the table was correct, the user need only supply the address of the entry point to the SVC routine. This would require the use of AMASPZAP. Using AMASPZAP it would be possible to update or replace all attributes stored in the SVC table entry.

During SYSGEN, if an SVC is not specified, the following type of entry will be generated:

IGCERROR	Entry Point Address
Type 2	
Unauthorized	
No locks needed	

If during subsequent operation of the system, an SVC by that number is issued, the routine IGCERROR will be entered. This will issue an ABEND for the issuing program.

Missing Interruption Handler

The missing interruption handler checks whether expected I/O interruptions occur within a specified period of time. If the interruptions do not occur, the operator is notified so that steps can be taken to correct the situation before system status is harmed.

The missing interruption handler checks for pending device ends, channel ends, DDR swaps, and MOUNT commands. When a pending condition is found, the condition is indicated in the UCB of the device. After the specified time elapses, another check is made for the pending condition. If the condition is still pending, a message is issued informing the operator what condition is pending and what operator action is required.

Establishing a Time Interval

The IBM-supplied CSECT IGFINTVL provides a time interval of 3 minutes. If the interval is not changed or is incorrectly modified, an interval of 3 minutes is assumed.

To change the time interval, specify the desired interval in a modification of the CSECT IGFINTVL as indicated in Figure 15.

```
//MODIFY     JOB     MSGLEVEL=(1,1)
//STEP       EXEC    PGM=AMASPZAP
//SYSPRINT   DD      SYSOUT=A
//SYSLIB     DD      DSNAME=SYS1.LPALIB,DISP=OLD
//SYSIN      DD      *
             NAME    IGFDIO  IGFINTVL
             VERIFY  0000  FOFOFOF3
             REP     0000  FOFOFXFX
```

Figure 15. Changing the Missing Interruption Handler Time Interval

In the REP statement in Figure 15, FXFX represents the time interval, and must be replaced within the range F0F1 to F9F9. If F0F0 is specified, the time interval defaults to 3 minutes.

If a new time interval is specified, it does not take effect immediately. The new time interval becomes effective only after the next IPL of the system, and only if the CLPA parameter is specified at the IPL.

Note: The CSECT IGFINTVL is eight bytes in length, and initially contains the character string C'00030000', where 3 indicates the time interval.

Adding Code to the Power Warning Feature Support

The Power Warning Feature Support, along with its supporting hardware prevents the loss of information in real storage at the occurrence of a utility power disturbance. The supporting hardware must include an Uninterruptable Power Supply to provide alternate power and equipment to signal the Power Warning Feature Support routines when a disturbance occurs.

The Power Warning Feature Support, after receiving the signal of a power disturbance, and determining the significance of the disturbance can transfer the contents of real storage to disk. After utility power is restored, the customer can use the Power Warning Feature Support 'restore' routine to refresh the contents of real storage from disk.

Adding code to the machine check handler appendage -- You can add code that will be executed after receipt of the signal that indicates that a sustained power disturbance has occurred. Your code can then permit transfer of real storage to a warn data set or have control returned to the supervisor. Insert your code in the machine check handler appendage ICFBDF00.

Adding code to the master scheduler initialization module -- You can add code that will execute when a warn data set contains an image of real storage and the system operator chooses FORM during system IPL. Your code will execute just before the warn data sets are erased and reformatted. Insert your code in the Power Warning Feature Support Initialization module ICFBIF00.

Note: For details on the warn data set, see **OS/VS2 System Programming Library: System Generation Reference,** GC26-3792.

Note: Your code must replace instructions, in either of the routines, that are bracketed with asterisks and specifically identified with comments. Since adding code to the Power Warning Feature Support requires considerable programming skill, before attempting any addition you should carefully examine the complexities involved.

Writing Code for the Machine Check Handler Appendage

You can insert code which will be executed when the Power Warning Feature Support is entered due to a power warning interrupt that would normally cause the transfer of real storage to the warn data set. After your code executes you can either cause control to be transferred to the dump routine or have control returned to the machine check handler for the system to continue processing.

As shown in Figure 16, the machine check handler appendage consists of three parts: 1) the warning appendage, 2) your code, and 3) the dump routine.

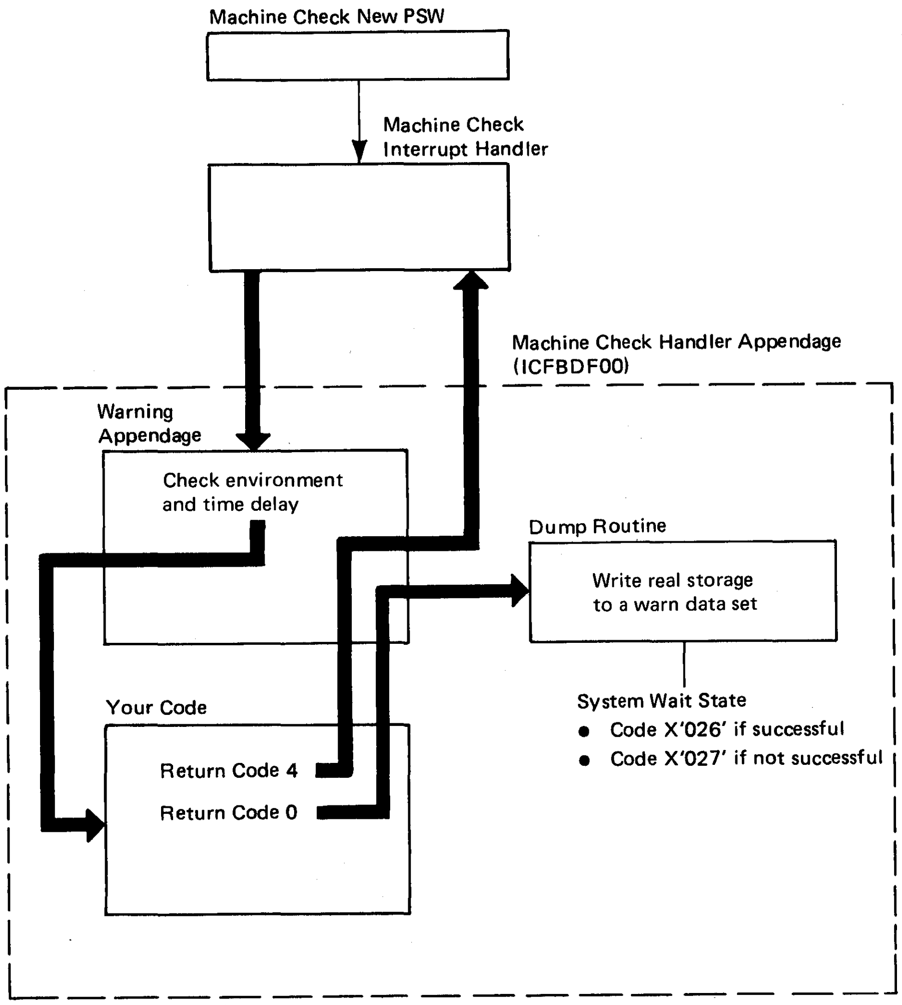
The warning appendage routine, after a power interruption, receives control from the system's machine check handler. This warning appendage monitors the power interruption during the time delay you specified in the CTRLPROG macro instruction at system generation. If a machine check occurs during the time delay, the remainder of the time delay is canceled.

Normally your code receives control at the end of the time delay, assuming the power interruption is still present. If the utility power returns before the expiration of the time delay, control returns to the supervisor, via the machine check handler.

The dump routine transfers the contents of real storage to a warn data set. After execution of the dump routine, the system enters a wait state.

To include you code:

- Obtain the source code for ICFBDF00.
- Replace the code that is bracketed with asterisks with your code.
- Reassemble the modified ICFBDF00.
- Re-linkedit the nucleus data set SYS1.NUCLEUS (IEANUC01), replacing the old ICFBDF00, with your modified one.
- Re-IPL your system.



Machine Check New PSW
Machine Check Interrupt Handler
Machine Check Handler Appendage (ICFBDF00)
Warning Appendage
Check environment and time delay
Dump Routine
Write real storage to a warn data set
System Wait State
• Code X'026' if successful
• Code X'027' if not successful
Your Code
Return Code 4
Return Code 0

Figure 16. Logical Placement of Your Code in the Machine Check Handler Appendage

Coding Considerations

At entry to your code, Register 9 addresses the PWF Communication area, Register 14 contains the return address, and Register 15 contains the address of your first instruction. Your code must save and restore all the general registers except register 15, prior to the exit from your code. You must also restore all control registers and all real storage outside of your inserted code. Register 15 must be set to 0 if you wish real storage to be transferred to a warn data set, or be set to 4 if you wish to resume system operations with all power warning interrupts disabled.

Your code will be entered:

- Key zero.
- Disabled for all interrupts.
- In supervisor state.

Your code must not:

- Use supervisor services.
- Contain Address Constants A-type or V-type since your code may be relocated.

Note: With a multiprocessing system, your program will be executed by only one CPU.

Writing Code for the Master Scheduler Initialization Routine

You can write a routine that will execute, when there is a real storage image on a warn data set, just before warn data sets are erased and reformatted.

As shown in Figure 17, during IPL, if the warn data set contains information from real storage, the system operator can choose to respond either REST or FORM. REST will cause real storage to be refreshed with the contents of the warn data set. FORM will cause a transfer of control to your code; after your code returns control, the warn data sets will be erased and reformatted by the Power Warning Feature Support. If you have not inserted any code, FORM will immediately cause the warn data set to be erased and reformatted.

To include you code:

1. Obtain the source code for ICFBIF00.
2. Replace the code bracketed with asterisks, with your code.
3. Reassemble the modified ICFBIF00.
4. Inserted your modified ICFBIF00 as follows:
   - If you have not generated your system -- replace the old ICFBIF00 in the SYS1.AOSCE data set on the distribution libraries with your modified ICFBIF00 and generate your system.
   - If you have generated your system -- re-linkedit your modified ICFBIF00 found in SYS1.AOSCE with the master scheduler initialization module found in SYSI.LINKLIB in the system library. The master scheduler initialization module is IEEMB860 for MVS.

Master Scheduler Initialization Routine

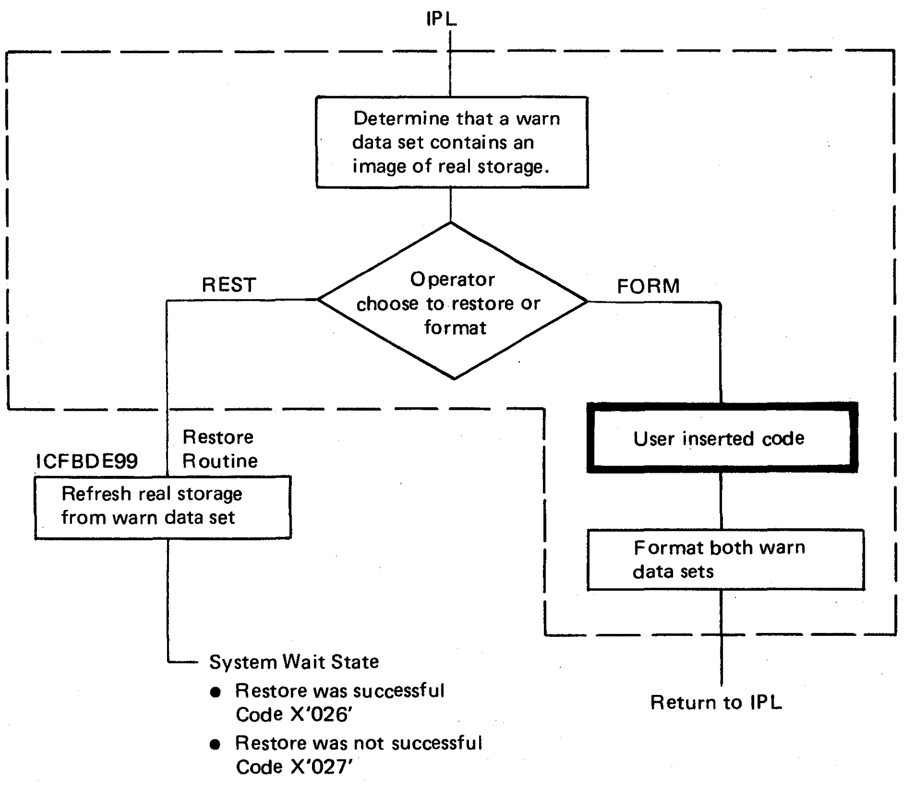

IPL
Determine that a warn data set contains an image of real storage.
REST
Operator choose to restore or format
FORM
ICFBDE99
Restore Routine
Refresh real storage from warn data set
User inserted code
Format both warn data sets
System Wait State
● Restore was successful Code X'026'
● Restore was not successful Code X'027'
Return to IPL

Figure 17. Logical Placement of Your Code in Master Scheduler Initialization Module

Coding Considerations:

At entry to your code: Register 13 addresses the register save area, Register 14 contains the return address, and Register 15 contains the address of your first instruction. Register 1 points to a word that contains the address of the TIOT for the warn data set.

If your code attempts to read information on the warn data set, it should first reference the control record to get vital information about the data set. The control record is the first record on logical cylinder 0 on each warn data set. This record indicates if the warn data set contains information from real storage and indicates its format. The format of the control record is shown in Figure 18.

When your code gets control the environment is as follows:

- Your code is pageable.
- Your code is transient.
- Your code cannot obtain permanent storage within the region.
- Consoles are available for your code to write to operator.
- Job Scheduler and SYSIN/SYSOUT services are not available.

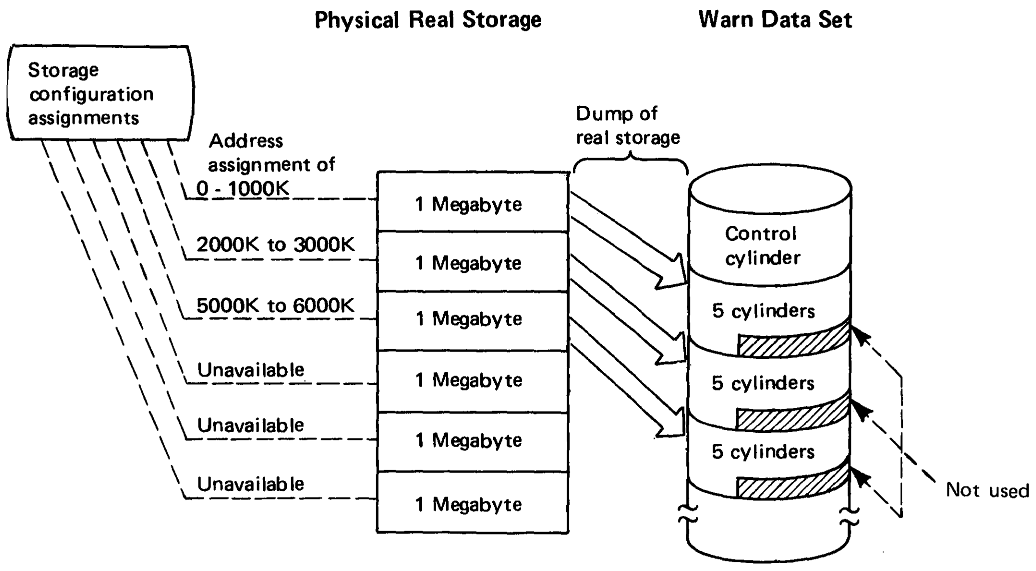
If your system is an M158 or M168 multiprocessor, you must consider that the first address after the last unused address starts a new cylinder on the warn data set. As shown in Figure 19, addresses 0, 2000K, and 5000K start new cylinders. An unused address refers to a location in real storage that is not available because of settings of the console switches.

Control Track Record

This record is located on cylinder 0 on each warn data set.

Offset	Size/Bits Length	Name	Description
0 (0)	4	ICFCTID	This is the identifier of the first word of the control track record. It always contains CNTL.
4 (4)	128	ICFCTCF	This area contains 128 byte-indicators. One indicator can be allocated to each of 128 cylinders. (128 cylinders will accomodate 16 megabytes of real storage.) Each byte indicator is structured as follows:
	00..		Indicates no data on this cylinder.
	01..		Real storage has been transferred to this cylinder and this cylinder contains no defecteve tracks.
	10..		Real storage has been transferred to this cylinder, and this cylinder contains a defective track.
	..xx xxxx		Contains the track number of either the defective or spare track. (The spare track is the last track on the cylinder.)
132(84)	1	ICFCTFLA	Status flags.
	00.. 00..		The data set is formatted, but contains no data.
	10.. 00..		This data set contains a successful transfer from real storage.
	00.. 10..		This data set contains a partial transfer from real storage.
	10.. 01..		This data set contains a successful transfer from real storage, but at least one track was found defective.
	..xx ..xx		Reserved. Set to zero.
133(85)	3	----	Reserved.
136(88)	4	ICFCTTS	Track size. Number of bytes in each track.
140(8C)	4	ICFCTAWA	The real storage address of the PWF Communication area in real storage.
144(90)	128	ICFCTB11	This area contains eight 16-byte fields. Each 16-byte field represents a contiguous area of real storage on this data set.
144(90)	16	ICFCTB11	Information concerning 1st contiguous area of real storage.
	4	ICFCTB11	Contains the real storage address of the 1st byte represented by this field. (In this case, this byte contains 0's.)
	4	ICFCTB12	Contains the cylinder and track on this data set where this contiguous real storage begins.
	4	ICFCTB13	Contains the real storage address of the last byte represented by this field.
	4	ICFCTB14	Contains the cylinder and track on this data set where this contiguous real storage ends.
160(AO)	16	ICFCTB21-24	Information concerning 2nd contiguous area of real storage.
176(BO)	16	ICFCTB31-34	Information concerning 3rd contiguous area of real storage.
192(CO)	16	ICFCTB41-44	Information concerning 4th contiguous area of real storage.
208(DO)	16	ICFCTB51-54	Information concerning 5th contiguous area of real storage.
224(EO)	16	ICFCTB61-64	Information concerning 6th contiguous area of real storage.
240(FO)	16	ICFCTB71-74	Information concerning 7th contiguous area of real storage.
256(100)	16	ICFCTB81-84	Information concerning 8th contiguous area of real storage.
272(110)	8	ICFCTST	Contains a true reading (binary) of the time-of-day clock when system was last IPLed: or if after a dump, the time at which processing of last PLD began.
280(118)	8	ICFCTED	Contains a true reading (binary) of time-of-day clock at the end of the last real storage transfer to the warn data set.
288(120)	4	ICFTTPC	Total number of tracks of each cylinder.
292(124)	4	ICFCTRDA	The address of this device. This field is set just before entering the restore routine.

Figure 18. Control Track Record



Note: Addresses 1000K to 2000K, and 3000K to 5000K are unused.
The first address after a unused address starts a new cylinder.

Figure 19. Storage Assignments on MP Systems

Limiting User Region Size - IEALIMIT

An installation can enforce a region-size limit by writing an exit routine that is invoked once per step when the initiator is establishing region size. If an installation-written exit routine does not exist, an IBM-supplied routine receives control.

The installation-written exit routine, which replaces the IBM-supplied routine, must be named IEALIMIT and must be link-edited into the nucleus. The routine must observe standard linkage conventions.

Upon entry to the IEALIMIT routine, the register contents are as follows:

Register	0	number of bytes requested by the application program for its region (specified explicitly through the REGION parameter or implicitly through the default JCL value)
Register	1	same as register 0
Register	13	address of standard save area
Register	14	return address
Register	15	entry point address for the IEALIMIT routine

Upon exit from the IEALIMIT routine, the register contents must be as follows:

Register 0	number of bytes to be used as the region parameter value
Register 1	number of bytes to be used as the limit on all types of requests from subpools 0-127, 251 and 252
Registers 2-13	restored
Registers 14-15	irrelevant

Note: The processing performed by IEALIMIT has no effect on the region size of a step that executes V=R. Such region sizes may be inspected and modified in the IEFUJV user exit as described in *OS/VS2 System Programming Library: System Management Facilities (SMF)*. The IEALIMIT routine may, however, terminate a V=R step by returning in register 1 a non-zero value less than that passed to it in register 0.

If the input register 1 is non-zero when the IBM-supplied IEALIMIT receives control, then IEALIMIT adds 64K to the contents of register 1 and returns. Register 0 remains unchanged. The register 1 value is used to limit the total allocation of storage from subpools 0-127, 251, and 252.

If the IBM-supplied IEALIMIT routine receives control and the input register 1 contains a zero, then IEALIMIT returns a zero in register 1 and no limit is assigned (to a job, a started program, or a TSO user). No limit is set only when the REGION parameter is not specified and the default value is zero.

When no limit is set, sufficient space within a region may be obtained via repeated small GETMAINs or via a single large GETMAIN, such that no space remains in the private area for use by the system. This situation is likely to occur when a variable GETMAIN is issued which specifies such a large maximum value that most or all of the space remaining in the private area is allocated to the requestor. Therefore, it is strongly recommended that a region size be specified on the JOB or EXEC statement, or that the default region size for the job class not be zero.

After the IEALIMIT routine determines the appropriate limit, it must pass back to IEAVPRT0, via register 1, a numeric value that represents the imposed limit in bytes. As noted above, a zero returned in register 1 indicates that a limit is not imposed. The IEALIMIT routine should pass back, in register 0, a value that is less than that in register 1. Both register 0 and 1 should be rounded to a multiple of 4K. IEAVPRT0 stores register 0 as the REGION parameter value and register 1 as the IEALIMIT value for future reference (that is, for use in processing subsequent GETMAINs as described below).

The REGION parameter value (register 0) should be less than the IEALIMIT value (register 1) to protect against programs that issue variable-length GETMAINS with very large maxima, and then do not immediately FREEMAIN part of that space or FREEMAIN such a small amount that a subsequent GETMAIN (possibly issued by a system service) causes the job to fail. As an example, suppose that a program issues a variable-length GETMAIN with a maximum of 2^{24}-1 bytes. If the REGION parameter value is not less than the IEALIMIT value, all the space in the region up to the IEALIMIT value will be allocated, and any subsequent GETMAIN that cannot be satisfied from free space in an already existing subpool will cause the job to fail. If however, the REGION parameter value is made less than the IEALIMIT value, only space up to the REGION parameter value will be allocated for the GETMAIN. Thus, an amount of space equal to the IEALIMIT value minus the REGION parameter value will remain for subsequent GETMAINs.

The REGION parameter value specifies the maximum amount of storage that can be allocated to a job by any single variable-length GETMAIN request. The IEALIMIT value specifies the maximum total storage that can be allocated to a job by any combination of GETMAINs. The relationship between the REGION parameter value and the IEALIMIT value and their effect upon both fixed-length and variable-length GETMAINs is shown in Figure 20.

Type of GETMAIN	Results
Fixed-length:	
IEALIMIT value minus currently alloc space $\geq$ request	Satisfied
IEALIMIT value minus currently alloc space $<$ request	Rejected
Variable-length: (REGION parm $<$ IEALIMIT and IEALIMIT=0)	or REGION parm $>$ IEALIMIT
REGION parm minus currently alloc space $\geq$ max	Maximum is allocated.
min $<$ REGION parm minus currently alloc space $<$ max	Unallocated amount is allocated.
REGION parm minus currently alloc space $\leq$ min	Minimum is allocated as long as IEALIMIT is not exceeded (in which case the request fails unless IEALIMIT=0).
Variable-length: (REGION parm $\geq$ IEALIMIT	and IEALIMIT $\neq$ 0)
IEALIMIT minus currently alloc space $\geq$ max	Maximum is allocated.
min $<$ IEALIMIT minus currently alloc space $<$ max	Unallocated amount is allocated.
IEALIMIT minus currently alloc space $\leq$ min	Minimum is allocated as long as IEALIMIT is not exceeded (in which case the request fails).

Figure 20. The Effects of IEALIMIT and REGION Values on Various GETMAINs

For example, assume that application program A has the following characteristics:

```
IEALIMIT value              150K
REGION-parameter value      100K
Currently allocated space    80K
```

Program A issues the following variable-length GETMAIN requests in the order indicated:

1. Request 5K-10K: 10K is allocated; currently allocated space is now 90K. Because the amount currently allocated (80K) does not exceed the REGION-parameter value (100K) and because the amount unallocated (20K - relative to the REGION-parameter value) is greater than the maximum amount requested (10K), the maximum is allocated.
2. Request 5K-100K: 10K is allocated; currently allocated space is now 100K. Because the amount unallocated (10K - relative to the REGION-parameter value) is between the minimum and maximum, the amount unallocated is allocated.
3. Request 40K-100K: 40K is allocated; currently allocated space is now 140K. Because the amount unallocated (0K - relative to the REGION-parameter value) is less than the minimum amount requested (40K), the minimum amount is allocated.
4. Request 15K-50K: the GETMAIN request fails. The amount unallocated (0K - relative to the REGION-parameter value) is less than the minimum amount requested (15K). If the minimum amount were allocated, the currently allocated amount would become 155K, which exceeds the IEALIMIT value (150K). Therefore, the request fails.

Branch Entering POST

Branch entry to POST provides all the normal ECB/RB POST processing. In general, the caller must hold the local lock and be in key zero, supervisor state. Upon completion of the POST process, control is given back to the caller in key zero, supervisor state, with the local lock held.

The following tables illustrate the POST function and the branch entry points through which those functions can be performed, input parameters to POST, and output parameters from POST.

	IEAOPT01	IEAOPT02
Local ECB POST	X	X
Local POST without ECB	X	X
Cross address space POST	X[1]	
[1]The local lock does not need to be held for a cross address space POST at this entry point.		

Figure 21. POST Function and Branch Entry Points

Registers	IEAOPT01	IEAOPT02
0		
1		
10	Completion code[1]	Completion code
11	ECB Address[2]	ECB Address
12	Erret Address[3]	
13	ASCB Address[3]	
14	Return Address	Return Address
15	Entry Point Address	Entry Point Address
	(CVTOPT01)	(CVTOPT02)

[1] If POST-without-ECB, contains RB address.
[2] If POST-without-ECB, set zero; if local address space POST, insure high-order byte of register is zero; if cross address space POST, set high-order byte of register to X'80'.
[3] Only necessary when performing cross address space POST.

Figure 22. POST Branch Entry Input

IEAOPT01[1]	Registers saved/restored - 0-9, 12,[2] 13,[2] 14 Registers not saved - 10-11, 15
IEAOPT02	Registers saved/restored - 0-9, 12-14 Registers not saved - 10, 11, 15

[1] If performing a cross address space POST and the local lock is not held, only registers 9 and 14 will be retained; all other register contents will be unpredictable.
[2] If performing a cross address space POST, will not be saved and restored; the contents will be unpredictable.

Figure 23. POST Branch Entry Output

Branch Entering WAIT

Branch entry to WAIT provides all the normal ECB/RB WAIT processing; this function is not available to Type 1 SVCs or SRBs. The caller must be in key zero, supervisor state, and hold the local lock. While holding the local lock and before branching to WAIT, the caller must establish the PSW and register return environment in his RB and TCB, respectively. When WAIT is invoked, only the local lock should be held by the caller. WAIT will:

- store the ECB/ECBLIST address into the register 1 location of the TCB register save area, (user data cannot be passed through this field/register).
- purge all FRRs.
- release the local lock.
- return control to the dispatcher (control is not returned to the caller even though the number of events to be waited on have already occurred.

WAIT can be branch entered without any ECBs identified. This results in the wait count in the current RB being set to the specified value. The corresponding POSTs-without-ECB would then activate the RB. Caution must be exercised to insure that the WAIT-without-ECB always preceeds the POST-without-ECB. Failure to follow this sequence will result in the RB waiting indefinately.

The parameters for branch entry to WAIT are as follows:

Register	0	Contains the WAIT count in the low-order byte. The high-order bit on indicates long-wait (LONG=YES).
Register	1	Contains the ECB pointer value. If only one ECB is being waited on, place that ECB address in register 1. If a list of ECBs is being waited on, place the complimented ECBLIST address in register 1. If the WAIT-without-ECB function is being requested, set register 1 to a value of zero.
Register	15	Contains the branch entry address to WAIT (IEAVWAIT) which is obtained from the CVT (CVTVWAIT).

Part II: Reference — Macro Instructions

You can communicate service requests to the control program using a set of macro instructions provided by IBM. Whereas most of the macro instructions have no restrictions on use by applications programmers some of the macro instructions should be restricted in use to systems programmers and installation-approved personnel.

This section describes those **Supervisor macro instructions** that should be installation-controlled. Some macro instructions should be totally restricted in use; these are described fully in this book. Other macro instructions are not restricted in use, but do contain some parameters that should be restricted; in these cases, only the parameters that should be restricted are fully described in this book. In all cases, however, the format of the complete macro instruction is described.

Figure 24 lists all macro instructions described in this book, and indicates which ones are fully described and which ones are partially described.

Macro Instruction	Fully Described	Partially Described
CALLDISP	X	
CALLRTM	X	
CIRB	X	
DSGNL	X	
EXTRACT	X	
MODESET	X	
NIL	X	
OIL	X	
PGFIX	X	
PGFREE	X	
PURGEDQ	X	
QEDIT	X	
RESERVE	X	
RISGNL	X	
RPSGNL	X	
SCHEDULE	X	
SDUMP	X	
SETFRR	X	
SETLOCK	X	
SPOST	X	
STAE	X	
SYNCH	X	
TESTAUTH	X	
ATTACH		X
DEQ		X
ENQ		X
ESTAE		X
EVENTS		X
FREEMAIN		X
GETMAIN		X
POST		X
SETRP		X
SPIE		X
STATUS		X
WTO		X
WTOR		X

Figure 24. Macro Instruction Coverage

The macro instructions are available only when programming in the assembler language, and are processed by the assembler program using macro definitions supplied by IBM and placed in the macro library when the system was generated. The processing of the macro instruction by the assembler program results in a macro expansion, generally consisting of data and executable instructions in the form of assembler language statements. The data fields are the parameters to be passed to the requested control program routine; the executable instructions generally consist of a branch around the data, instructions to load registers, and either a branch instruction or a supervisor call (SVC) to give control to the proper program. The exact macro expansion appears as part of the assembler output assembler output listing.

Macro Instruction Forms

When written in the standard form, some of the macro instructions result in instructions that store into an inline parameter list. The option of storing into an out-of-line parameter list is provided to allow the use of these macro instructions in a reenterable program. You can request this option through the use of list and execute forms. When list and execute forms exist for a macro instruction, their descriptions follow the description of the standard form.

Use the list form of the macro instruction to provide a parameter list to be passed either to the control program or to a problem program, depending on the macro instruction. The expansion of the list form contains no executable instructions; therefore registers cannot be used in the list form.

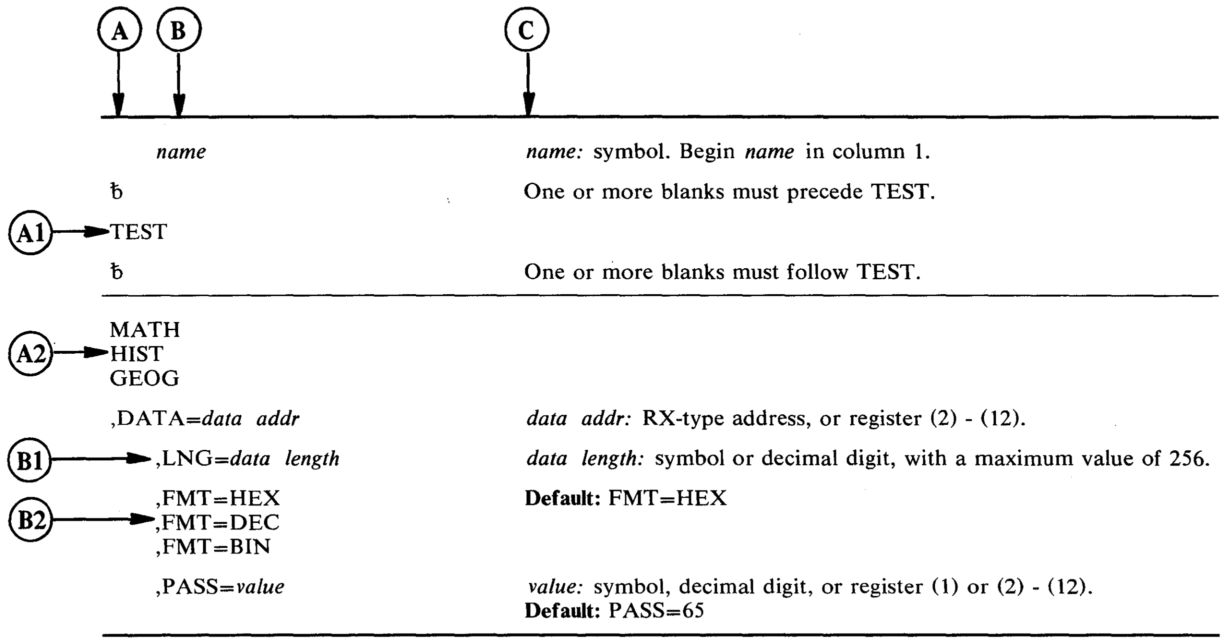
Use the execute form of the macro instruction in conjunction with one or two parameter lists established using the list form. The expansion of the execute form provides the executable instructions required to modify the parameter lists and to pass control to the required program.

The descriptions of the following macro instructions assume that the standard begin, end, and continue columns are used -- for example, column 1 is assumed as the begin column. To change the begin, end, and continue columns, code the ICTL instruction to establish the coding format you wish to use. If you do not use ICTL, the assembler recognizes the standard columns. To code the ICTL instruction, see **OS/VS - DOS/VS - VM/370 Assembler Language.**

Coding the Macro Instructions

The table appearing near the beginning of each macro instruction indicates how the macro instruction is to be coded. The table does not attempt to explain the meanings of the parameters; the parameters are explained following the table.

Figure 25 presents a sample macro instruction, TEST, and summarizes all the coding information that is available for it. The table is divided into three columns.



name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede TEST.
TEST	
ƀ	One or more blanks must follow TEST.
MATH HIST GEOG	
,DATA=*data addr*	*data addr:* RX-type address, or register (2) - (12).
,LNG=*data length*	*data length:* symbol or decimal digit, with a maximum value of 256.
,FMT=HEX ,FMT=DEC ,FMT=BIN	**Default:** FMT=HEX
,PASS=*value*	*value:* symbol, decimal digit, or register (1) or (2) - (12). **Default:** PASS=65

Figure 25. Sample Macro Instruction

- The first column,(A), contains those parameters that are required for that macro instruction. If a single line appears in that column,(A1), the parameter on that line is required and must be coded. If two or more lines appear together,(A2), the parameter appearing on one and only one of the lines must be coded.
- The second column,(B), contains those parameters that are optional for that macro instruction. If a single line appears in that column,(B1), the parameter on that line is optional. If two or more lines appear together,(B2), although the entire parameter is optional, if you elect to make an entry, one and only one of the lines may be coded.
- The third column,(C), provides additional information for coding the macro instruction. When substitution of a variable is required, the following classifications should be understood:

symbol: any symbol valid in the assembler language. That is, an alphabetic character followed by 0-7 alphameric characters, with no special characters and no blanks.

decimal digit: any decimal digit up to the value indicated in the parameter description. If both symbol and decimal digit are indicated, an absolute expression is also allowed.

register (2) - (12): one of general registers 2 through 12, specified within parentheses, previously loaded with the right-adjusted value or address indicated in the parameter description. The unused high-order bits must be set to zero. The register may be designated symbolically or with an absolute expression.

register (0): general register 0, previously loaded as indicated under register (2) - (12) above. Designate the register as (0) only.

register (1): general register 1, previously loaded as indicated under register (2) - (12) above. Designate the register as (1) only.

RX-type address: any address that is valid in an RX-type instruction (for example, LA).

A-type address: any address that may be written in an A-type address constant.

default: a value that is used in default of a specified value, and that is assumed if the parameter is not coded.

Use the parameters to specify the services and options to be performed, and write them according to the following general rules:

- If the selected parameter is written in all capital letters (for example, STEP, DUMP, or RET=USE), code the parameter exactly as shown.
- If the selected parameter is written in italics (for example, *value* or *comp code*), substitute the indicated value, address, or name.
- If the selected parameter is a combination of capital letters and italics separated by an equal sign (for example, EP=*entry point*), code the capital letters and equal sign as shown, and then make the indicated substitution for the italics.
- Read the table from top to bottom.
- Code commas and parentheses exactly as shown.
- Positional parameters (parameters without equal signs) appear first and must be coded in the order shown. Keyword parameters (parameters with equal signs) may be coded in any order.
- If a parameter is selected, read column 3 before proceeding to the next parameter. Column 3 will often contain notes pertaining to restrictions on coding the parameter.

Continuation Lines

You can continue the parameter field of a macro instruction on one or more additional lines according to the following rules:

1. Enter a continuation character (not blank, and not part of the parameter coding) in column 72 of the line.
2. Continue the parameter field on the next line, starting in column 16. All columns to the left of column 16 must be blank.

You can code the parameter field being continued in one of two ways. Code the parameter field through column 71, with no blanks, and continue in column 16 of the next line; or truncate the parameter field by a comma, where a comma normally falls, with at least one blank before column 71, and then continue in column 16 of the next line. Figure 26 shows an example of each method.

```
NAME1       OP1     OPERAND1,OPERAND2,OPERAND3,OPERAND4,OPERAX
                    ND5,OPERAND6             THIS IS ONE WAY
NAME2       OP2     OPERAND1,OPERAND2,       THIS IS ANOTHER  X
                    OPERAND3,                WAY              X
                    OPERAND4
```

Figure 26. Continuation Coding

ATTACH — Create a New Task

The ATTACH macro instruction is described in the **OS/VS2 Supervisor Services and Macro Instructions**, with the exception of the JSTCB, SM, SVAREA, KEY, DISP, JSCB, TID, NSHSPV, and NSHSPL parameters. These parameters are restricted in use and should only be used with tasks in supervisor mode, having a system protection key. If they are used with other tasks, the default values are used.

The syntax of the complete ATTACH macro instruction is shown below. However, only the explanation of the restricted parameters is presented. Explanation of the other parameters can be found in **OS/VS2 Supervisor Services and Macro Instructions.**

The standard form of the ATTACH macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ATTACH.
ATTACH	
ƀ	One or more blanks must follow ATTACH.

EP=*entry name*
EPLOC=*entry name addr*
DE=*list entry addr*
	entry name: symbol.
	entry name addr: A-type address, or register (2) - (12).
	list entry addr: A-type address, or register (2) - (12).

,DCB=*dcb addr*	*dcb addr:* A-type address, or register (2) - (12).
,LPMOD=*limit prior nmbr*	*limit prior nmbr:* symbol, decimal digit, or register (2) - (12).
,DPMOD=*disp prior nmbr*	*disp prior nmbr:* symbol, decimal digit, or register (2) - (12).
,PARAM=(*addr*) ,PARAM=(*addr*),VL=1	*addr:* A-type address, or register (2) - (12). **Note:** *addr* is one or more addresses, separated by commas. For example, PARAM=(*addr,addr,addr*)
,ECB=*ecb addr*	*ecb addr:* A-type address, or register (2) - (12).
,ETXR=*exit rtn addr*	*exit rtn addr:* A-type address, or register (2) - (12).
,GSPV=*subpool nmbr* ,GSPL=*subpool list addr*	*subpool nmbr:* symbol, decimal digit, or register (2) - (12). *subpool list addr:* A-type address, or register (2) - (12).
,SHSPV=*subpool nmbr* ,SHSPL=*subpool list addr*	*subpool nmbr:* symbol, decimal digit, or register (2) - (12). *subpool list addr:* A-type address, or register (2) - (12).
,SZERO=YES ,SZERO=NO	**Default:** SZERO=YES
,TASKLIB=*dcb addr*	*dcb addr:* A-type address, or register (2) - (12).
,STAI=(*exit addr*) ,STAI=(*exit addr,parm addr*) ,ESTAI=(*exit addr*) ,ESTAI=(*exit addr,parm addr*)	*exit addr:* A-type address, or register (2) - (12). *parm addr:* A-type address, or register (2) - (12).
,PURGE=QUIESCE ,PURGE=NONE ,PURGE=HALT	**Note:** PURGE may be specified only if STAI or ESTAI is specified. **Default for STAI:** PURGE=QUIESCE **Default for ESTAI:** PURGE=NONE
,ASYNCH=NO ,ASYNCH=YES	**Note:** ASYNCH may be coded only if STAI or ESTAI is specified. **Default for STAI:** ASYNCH=NO **Default for ESTAI:** ASYNCH=YES
,TERM=NO ,TERM=YES	**Note:** TERM may be specified only if ESTAI is specified. **Default:** TERM=NO
,JSTCB=NO ,JSTCB=YES	**Default:** JSTCB=NO
,SM=PROB ,SM=SUPV	**Default:** SM=PROB
,SVAREA=YES ,SVAREA=NO	**Default:** SVAREA=YES
,KEY=PROP ,KEY=ZERO	**Default:** KEY=PROP
,DISP=YES ,DISP=NO	**Default:** DISP=YES
,JSCB=*jscb addr*	*jscb addr:* A-type address, or register (2) - (12).
,TID=*task id*	*task id:* decimal digits 0-255, or register (2) - (12). **Default:** TID=0
,NSHSPV=*subpool nmbr* ,NSHSPL=*subpool list addr*	*subpool nmbr:* symbol, decimal digit, or register (2) - (12). *subpool list addr:* A-type address, or register (2) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters restricted in use are explained below. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

,JSTCB = NO
,JSTCB = YES
specifies whether the attached task is a new job step (YES) or a task in the present job step (NO). If YES is specified, the address of the TCB of the newly created task is placed in the TCBJSTCB field of the TCB; if NO is specified, the TCBJSTCB field of the task using ATTACH is propagated to the new task.

,SM = PROB
,SM = SUPV
specifies that the system is to run in problem program mode (PROB) or in supervisor mode (SUPV) when executing the attached task.

,SVAREA = YES
,SVAREA = NO
specifies whether a save area is needed for the attaching task. If YES is specified, the ATTACH routine will obtain a 72-byte save area. If both attaching and attached task share subpool zero, the save area is obtained there; otherwise, it is obtained from a new 4K-byte block.

,KEY = PROP
,KEY = ZERO
specifies whether the protection key of the newly created task should be zero (ZERO) or copied from the TCBPKF field of the TCB for the task using ATTACH (PROP).

,DISP = YES
,DISP = NO
specifies whether the subtask is to be dispatchable (YES) or nondispatchable (NO). (Note: If DISP=NO is specified, the attaching task must use the STATUS macro instruction to reset the TCBANDSP nondispatchability bit to 0, before the ATTACH processing can be completed for the new task.)

,JSCB = *jscb addr*
specifies the address of the job step control block. If specified, the JSCB is used for the new task. Otherwise, the JSCB of the attaching task is also used for the new task.

,TID = *task id*
specifies the task identifier to be placed in the TCBTID field of the attached task.

,NSHSPV = *subpool nmbr*
,NSHSPL = *subpool list addr*
specifies the virtual storage subpool number 236 or 237, or the address of a list of virtual storage subpool numbers 236 and 237. The subpools specified will not be shared with the subtask.

If NSHSPL is specified, the first byte of the list contains the number of bytes remaining in the list; each of the following bytes contains a virtual storage subpool number.

,RELATED = *value*
specifies information used to self-document macro instructions by 'relating' functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

The RELATED parameter is available on macro instructions that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE, and on macro instructions that relate to previous occurrences of the same macro instructions (for example, CHAP and ESTAE).

The parameter may be used, for example, as follows:

```
GET1    GETMAIN     R,LV=4096,RELATED=(FREE1,'GET STORAGE')
FREE1   FREEMAIN    R,LV=4096,A=(1),RELATED=(GET1,'FREE STORAGE')
```

When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful completion.
04	ATTACH was issued in a STAE exit; processing not completed.
08	Insufficient storage available for control block for STAI/ESTAI request; processing not completed.
0C	Invalid exit routine address or invalid parameter list address specified with STAI parameter; processing not completed.
14	Authorized task specifying JSTCB=YES was not itself a job step task; processing not completed.
18	Attempt to create a new subtask would result in both job step tasks and non-job step tasks being subtasks of current task; processing not completed.

Note: For any return code other than 00, register 1 is set to zero upon return.

Note: The program manager processing for ATTACH is performed under the new subtask, after control has been returned to the originating task. Therefore, it is possible for the originating task to obtain return code 00, and still not have the subtask successfully created (for example, if the entry name could not be found by the program manager). In such cases, the new subtask is abnormally terminated.

Ignored Parameters

The following parameters, available with Release 1 of OS/VS2, are ignored if coded in MVS:

GIVEJPQ=YES
GIVEJPQ=NO
TSLOGON=YES
TSLOGON=NO
LSQA=n

Example 1

Operation: Attach program SYSPROGM, which will run with protection key 0 and in supervisor mode. Subpool 0 is not to be shared, and the new task is not to have a savearea provided.

```
ATTACH  EP=SYSPROGM,KEY=ZERO,SM=SUPV,SZERO=NO,SVAREA=NO
```

Example 2

Operation: Attach as a new job step the program name addressed in register 7. The new task is to run in problem program mode, a savearea is to be provided, a job step control block is provided, subpool 0 is not to be shared, a task library DCB is provided, and the new task is to be nondispatchable.

```
ATTACH  EPLOC=(7),SM=PROB,JSTCB=YES,SVAREA=YES,SZERO=NO,
        JSCB=(5),DISP=NO,TASKLIB=(8)
```

CALLDISP — Force Dispatcher Entry

The CALLDISP macro instruction expands into an SVC that results in the caller's status being saved in the current TCB/RB, then the dispatcher is entered. The dispatcher then searches for the highest priority ready work to dispatch. When this task is redispatched, control is returned to the next sequential instruction.

The CALLDISP macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CALLDISP.
CALLDISP	
ƀ	One or more blanks must follow CALLDISP.

Note: CALLDISP contains no optional or required parameters.

Example 1

Operation: Pass control to a higher priority ready task.

CALLRTM — Call Recovery/Termination Manager

The CALLRTM macro instruction is used to direct the services of the recovery/termination manager to a task or address space other than itself or its callers. The recovery/termination manager selects the appropriate recovery or termination process according to the status of the system and the requests of its invokers.

CALLRTM may be used only by key 0 supervisor state routines. After execution of the macro instruction, control is returned to the caller.

The CALLRTM macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CALLRTM.
CALLRTM	
ƀ	One or more blanks must follow CALLRTM.

TYPE=ABTERM
TYPE=MEMTERM	

,COMPCOD=*comp code*	*comp code:* symbol, decimal digit, or register (2) - (12).
,ASID=*ASID addr*	*ASID addr:* decimal digits 0 - 32,765 or register (2) - (15).
,TCB=*tcb addr*	*tcb addr:* 0, or register (0) or (2) - (12). **Note:** This parameter may only be specified with TYPE=ABTERM.
,DUMP=YES ,DUMP=NO	**Default:** DUMP=YES **Note:** This parameter may only be specified with TYPE=ABTERM.
,STEP=NO ,STEP=YES	**Default:** STEP=NO **Note:** This parameter may only be specified with TYPE=ABTERM.
,DUMPOPT=*parm list addr*	*parm list addr:* register (3) - (15).

The parameters are explained below:

TYPE = ABTERM
TYPE = MEMTERM
specifies that the services of the recovery/termination manager is being directed towards another task (ABTERM) or that an address space is to be terminated (MEMTERM). For MEMTERM, all recovery processing in the address space is circumvented.

,COMPCOD = *comp code*
specifies the completion code associated with the abnormal termination. This parameter can be specified as a hexadecimal code (x'80A'), a decimal code (2058), or a register containing a hexadecimal code; in all cases, the result is hexadecimal.

,ASID = *ASID addr*
specifies the address space identifier of the address space to be terminated (for MEMTERM) or the address space identifier containing the TCB of the task to be terminated (for ABTERM).

,TCB = *tcb addr*
specifies the TCB address of the task to be terminated.

,DUMP=YES
,DUMP=NO
specifies that a dump is (YES) or is not (NO) to be taken. If the DUMPOPT parameter is not also specified, the contents of the dump is defined by the //SYSABEND or //SYSDUMP DD statement and the system or user defined defaults.

,STEP=NO
,STEP=YES
specifies that the entire job step is (YES) or is not (NO) to be abnormally terminated.

,DUMPOPT=*parm list addr*
specifies the address of a parameter list valid for the SNAP macro instruction. The parameter list is used to produce a tailored dump, and may be created by using the list form of the SNAP macro instruction, or a compatible list may be created. The TCB and DCB options available on SNAP will be ignored if they appear in the parameter list; the TCB used will be for the task that received ABEND, the DCB used will be provided by the ABDUMP routine. If a //SYSABEND or //SYSDUMP DD statement is not provided, the DUMPOPT parameter is ignored.

Example 1

Operation: Terminate the current address space with a completion code of 123.

```
CALLRTM     TYPE=MEMTERM,COMPCOD=123,ASID=0
```

Example 2

Operation: Schedule ABTERM of the TCB whose address is specified in register 8. The ABTERM of this TCB will take place in the address space identified by the ASID specified in register 5, and will have a completion code of 123.

```
CALLRTM     TYPE=ABTERM,COMPCOD=123,ASID=(5),TCB=(8)
```

CIRB — Create Interruption Request Block

The CIRB macro instruction is included in SYS1.MACLIB and must be included in the system at system generation time if the macro instruction is to be used. The issuing of this macro instruction causes a supervisor routine (called the exit effector routine) to create an interruption request block (IRB). In addition, other parameters of this macro instruction may specify the building of a register save area and/or a work area to contain interruption queue elements, which are used by supervisor routines in the scheduling of the execution of user exit routines.

The CIRB macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CIRB.
CIRB	
ƀ	One or more blanks must follow CIRB.
EP=*entry name addr*	*entry name addr:* RX-type address, or register (0) or (2) - (12).
,KEY=PP ,KEY=SUPR	**Default:** KEY=PP
,MODE=PP ,MODE=SUPR	**Default:** MODE=PP
,SVAREA=NO ,SVAREA=YES	**Default:** SVAREA=NO
,RETIQE=YES ,RETIQE=NO	**Default:** RETIQE=YES
,STAB=(DYN)	
,WKAREA=*workarea size*	*workarea size:* Decimal digit, or register (2) - (12). **Default:** zero
,BRANCH=NO ,BRANCH=YES	**Default:** BRANCH=NO
,RETRN=NO ,RETRN=YES	**Default:** RETRN=NO **Note:** This parameter has meaning only if RETIQE=NO is specfied above.

The parameters are explained below:

EP=*entry name addr*
specifies the address of the entry name of the user's asynchronous exit routine.

,KEY=PP
,KEY=SUPR
specifies whether the asynchronous exit routine will operate with a key of zero (SUPR) or with a key obtained from the TCB of the task issuing the CIRB macro instruction (PP).

,MODE=PP
,MODE=SUPR
specifies whether the asynchronous exit routine will be executed in problem program (PP) or supervisor (SUPR) mode.

,SVAREA = NO
,SVAREA = YES
specifies whether a 72-byte register save area is to be obtained from the virtual storage assigned to the problem program. If a save area is requested, CIRB places the save area address in the IRB. The address of this area is passed to the user routine via register 13.

,RETIQE = YES
,RETIQE = NO
specifies whether the associated queue elements are request queue elements (YES) or interruption queue elements (NO).

,STAB = (DYN)
specifies that the IRB (including the work area) will be freed by EXIT.

Branch Entry Interface

For BRANCH=YES, the branch entry interface is as follows:

- The caller must be in supervisor state, key 0, and own the local lock and no locks above the SALLOC in the locking hierarchy.
- The caller must pass a TCB address in register 4 to be used by GETMAIN when allocating space for the IRB and for the problem program save area. Also, if a problem key is specified in the KEY= parameter of the CIRB, the TCBPKF field of that TCB will be used.
- Upon return, register 1 will contain the address of the created IRB, registers 0, and 2-14 will be unchanged, and register 15 will be unpredictable.
- Control will be returned in supervisor state, key 0, with the same locks held as on entry.

Note: If the STAB parameter is omitted from the CIRB macro instruction, the IRB will remain available for later use by the task issuing the macro.

,WKAREA = *workarea size*
specifies the size, in doublewords, of the work area to be included in the IRB. The area may be used to build IQE's. The maximum size is 255 double words.

,BRANCH = NO
,BRANCH = YES
specifies that branch linkage (YES) or SVC linkage (NO) to CIRB will be provided.

,RETRN = NO
,RETRN = YES
specifies that the IQE will (YES) or will not (NO) be returned to the available queue when the asynchronous exit terminates.

Ignored Parameters

The following parameters, available with Release 1 of OS/VS2, are ignored if coded in MVS:

TYPE = IRB
ENABLE = YES
STAB = (RE)

The following parameters are no longer acceptable:

TYPE = TIRB
ENABLE = NO

Example 1

Operation: Create an IRB to be used in scheduling an asynchronous exit. The exit will be scheduled via the IQE interface to Stage 2 Exit Effector, and will receive control in the supervisor state. The IRB will be freed when it terminates. The exit will receive control at the IQERTN label.

```
     CIRB     EP=IQERTN,MODE=SUPR,RETIQE=NO,STAB=(DYN),BRANCH=NO
```

Example 2

Operation: Create an IRB to be used in scheduling an asynchronous exit. The RQE interface to Stage 2 Exit Effector will be used to schedule the routine. The exit will get control at the RQETEST label.

```
     CIRB     EP=RQETEST,KEY=SUPR,MODE=SUPR,STAB=(DYN),BRANCH=NO
```

DEQ — Release a Serially Reusable Resource

The DEQ macro instruction is described in the **OS/VS2 Supervisor Services and Macro Instructions**, with the exception of the RMC, GENERIC, TCB, and UCB parameters. These parameters are restricted in use and should only be used with tasks that are authorized. The UCB parameter is used to release a device that was reserved with a RESERVE macro instruction.

The syntax of the complete DEQ macro instruction is shown below. However, only the explanation of the restricted parameters is presented. Explanation of the other parameters can be found in **OS/VS2 Supervisor Services and Macro Instructions**.

The standard form of the DEQ macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede DEQ.
DEQ	
ƀ	One or more blanks must follow DEQ.
(	
qname addr	*qname addr:* A-type address, or register (2) - (12).
,rname addr	*rname addr:* A-type address, or register (2) - (12).
, *,rname length*	*rname length:* symbol, decimal digit, or register (2) - (12). **Note:** *rname length* must coded if a register is specified for *rname addr*.
, ,STEP ,SYSTEM ,SYSTEMS	**Default:** STEP
,var1234	*var1234:* The preceding 4 parameters may be repeated up to 65,535 times.
)	
,RET=HAVE .RET=NONE	
,RMC=NONE ,RMC=STEP ,GENERIC=NO ,GENERIC=YES	**Default:** RMC=NONE **Default:** GENERIC=NO **Note:** If GENERIC=YES is specified, you must also specify RET=HAVE above.
,TCB=*tcb addr*	*tcb addr:* A-type address, or register (2) - (12). **Note:** TCB cannot be specified with RMC above.
,UCB=*ucb addr*	*ucb addr:* RX-type address, or register (2) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters restricted in use are explained below. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions**.

,RMC = NONE
,RMC = STEP
,GENERIC = NO
,GENERIC = YES
specifies optional parameters available to the system programmer:

RMC specifies that the reset-must-complete function is not to be used (NONE) or that the requesting task is to release the resources and terminate the must complete function (STEP). The NONE or STEP subparameter must agree with the subparameter specified in the SMC parameter of the corresponding ENQ macro instruction.

GENERIC specifies whether or not (YES or NO) all queue elements for the task under the specified major name will be dequeued, regardless of whether they have control of the resource.

,TCB=*tcb addr*
specifies the address of a fullword on a fullword boundary that contains the address of a TCB on whose behalf the DEQ is to be done. The caller (not the directed task) will be abnormally terminated if the RET parameter is omitted and an attempt is made to DEQ a resource not requested or not owned by the directed task.

,UCB=*ucb addr*
specifies the address of a fullword that contains the address of a UCB for a reserved device that is now being released.

Return codes are provided by the control program only if RET=HAVE is designated. If all of the return codes for the resources named in DEQ are 0 register 15 contains 0. If any of the return codes are not 0 register 15 contains the address of a virtual storage area containing the return codes as shown in Figure 27.

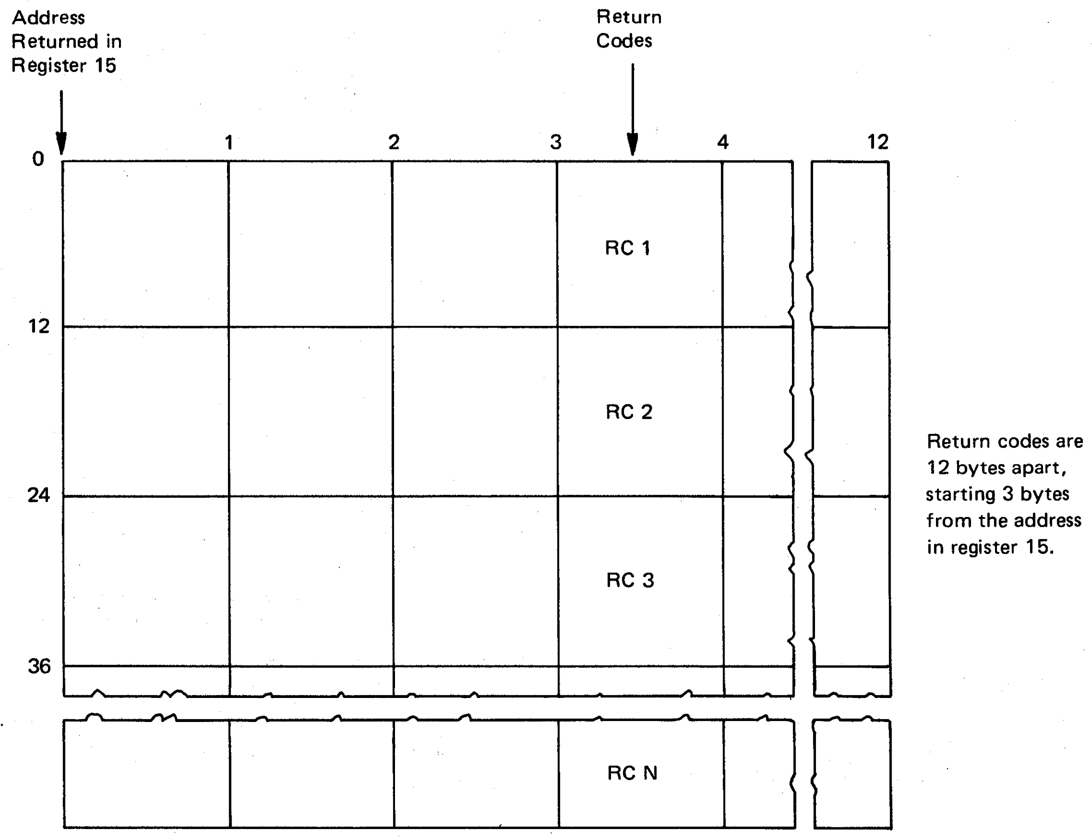

Address Returned in Register 15
Return Codes
0
1
2
3
4
12
RC 1
12
RC 2
24
RC 3
36
RC N
Return codes are 12 bytes apart, starting 3 bytes from the address in register 15.

Figure 27. Return Code Area Used by DEQ

The return codes are placed in the parameter list resulting from the macro expansion in the same sequence as the resource names in the DEQ macro instruction. The return codes are shown below.

Hexadecimal Code	Meaning
0	The resource has been released.
4	The resource has been requested for the task, but the task has not been assigned control. The task is not removed from the wait condition. (This return code could result if DEQ is issued within an exit routine which was given control because of an interruption.)
8	Control of the resource has not been requested by the active task, or the resource has already been released.

Example 1

Operation: Unconditionally release control of the resource in Example 1 of ENQ, and reset the 'must-complete' state.

```
DEQ (MAJOR1,MINOR1,8,STEP),RMC=STEP
```

Example 2

Operation: Conditionally release control of the resource in Example 2 of ENQ.

```
DEQ (MAJOR2,MINOR2,4,SYSTEM),TCB=(R2),RET=HAVE
```

Example 3

Operation: Unconditionally release control of the resource (device) in Example 1 of RESERVE.

```
DEQ (MAJOR3,MINOR3,,SYSTEMS),UCB=(R3)
```

DEQ (List Form)

The list form of the DEQ macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede DEQ.
DEQ	
ƀ	One or more blanks must follow DEQ.
(	
qname addr	*qname addr:* A-type address.
, ,*rname addr*	*rname addr:* A-type address.
, ,*rname length*	*rname length:* symbol or decimal digit.
, ,STEP ,SYSTEM ,SYSTEMS	**Default:** STEP
,*var1234*	*var1234:* The preceding 4 parameters may be repeated up to 65,535 times.
)	
,RET=HAVE ,RET=NONE	**Default:** RET=NONE
,RMC=NONE ,RMC=STEP	**Default:** RMC=NONE
,GENERIC=NO ,GENERIC=YES	**Default:** GENERIC=NO **Note:** If GENERIC=YES is specified, you must also specify RET=HAVE above.
,TCB=*tcb addr*	*tcb addr:* A-type address. **Note:** TCB cannot be specified with RMC above, and must be specified on the list form if used on the execute form.
,UCB=*ucb addr*	*ucb addr:* A-type address.
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

The parameters restricted in use are explained under the standard form of the DEQ macro instruction. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

DEQ (Execute Form)

The execute form of the DEQ macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede DEQ.
DEQ	
ƀ	One or more blanks must follow DEQ.
(	**Note:** (and) are the beginning and end of a parameter list. The entire list is optional. If nothing in the list is desired, then (,), and all parameters between (and) should not be specified. If something in the list is desired, then (,), and all parameters in the list should be specified as indicated at the left.
qname addr	*qname addr:* RX-type address, or register (2) - (12).
, *,rname addr*	*rname addr:* RX-type address, or register (2) - (12).
, *,rname length*	*rname length:* symbol, decimal digit, or register (2) - (12).
, ,STEP ,SYSTEM ,SYSTEMS	
,var1234	*var1234:* The preceding 4 parameters may be repeated up to 65,535 times.
)	**Note:** See note opposite (above.
,RET=HAVE ,RET=NONE	
,RMC=NONE ,RMC=STEP ,GENERIC=NO ,GENERIC=YES	**Note:** If GENERIC=YES is specified, you must also specify RET=HAVE above.
,TCB=*tcb addr*	*tcb addr:* RX-type address, or register (2) - (12). **Note:** TCB cannot be specified with RMC above, and must be specified on the execute form if used on the list form.
,UCB=*ucb addr*	*ucb addr:* RX-type address, or register (2) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E *,ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

The parameters restricted in use are explained under the standard form of the DEQ macro instruction. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

DSGNL — Issue Direct Signal

The DSGNL macro instruction uses the signal processor (SIGP) instruction to modify or sense the physical state of one of the CPUs in a tightly coupled multiprocessing system. Ten of the twelve SIGP hardware functions are defined as direct services and are accessible via the DSGNL macro instruction. The other two SIGP functions are accessible via the RISGNL and RPSGNL macro instructions.

The DSGNL macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede DSGNL.
DSGNL	
ƀ	One or more blanks must follow DSGNL.
SENSE START STOP RESTART IPR PR SSS IMPL ICPUR CPUR (0)	
,CPU=*PCCA addr*	*PCCA addr:* RX-type address, or register (1).

The parameters are explained below:

SENSE
START
STOP
RESTART
IPR
PR
SSS
IMPL
ICPUR
CPUR
(0)

specifies the action to be performed. If (0) is specified, the code indicating the desired function has already been loaded into bits 24-31 of register 0. The actions and codes are:

	Code	**Action**
SENSE	01	State of specified CPU is to be sensed
START	04	Start function
STOP	05	Stop function
RESTART	06	Restart function
IPR	07	Initial program reset function
PR	08	Program reset function
SSS	09	Stop and store status function
IMPL	0A	Initial microprogram load function
ICPUR	0B	Initial CPU reset function
CPUR	0C	CPU reset function

Note: Codes 0A, 0B, and 0C are only valid on a Model 168.

,CPU=*PCCA addr*

specifies the address of the physical configuration communication area (PCCA) of the CPU on which the function is to be executed.

When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	**Meaning**
00	Function successfully initiated, but not necessarily completed.
04	Function not completed because access path to the addressed processor was busy or the addressed processor was in a state where it could not accept and respond to the function code.
08	Function unsuccessful. Status returned in register 0.
12	Specified CPU is either not installed, not configured into the system, or powered off.
16	CPU is a uniprocessor and does not have signal processor sending and receiving capabilitites.

With a return code of 8, register 0 contains:

Bits	**Meaning**
0	Equipment check
1-23	Reserved
24	External call pending
25	Stopped
26	Operator intervening
27	Check stop
28	Not ready
29	Reserved
30	Invalid function
31	Receiver check

Example 1

Operation: The state of the CPU whose PCCA is located at PCCA is requested. If the CPU is executing or is in a wait state, a return code of 0 in register 15 will be provided; otherwise, a return code of 8 with status indicators in register 0 will be returned.

```
DSGNL    SENSE,CPU=PCCA
```

Example 2

Operation: The CPU whose PCCA address is in register 1 will be placed in the STOP state.

```
DSGNL    STOP,CPU=( 1 )
```

ENQ — Request Control of a Serially Reusable Resource

The ENQ macro instruction is described in the **OS/VS2 Supervisor Services and Macro Instructions**, with the exception of the SMC, ECB, and TCB parameters. These parameters are restricted in use and should only be used with tasks that are authorized.

The syntax of the complete ENQ macro instruction is shown below. However, only the explanation of the restricted parameters is presented. Explanation of the other parameters can be found in **OS/VS2 Supervisor Services and Macro Instructions.**

The standard form of the ENQ macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b̸	One or more blanks must precede ENQ.
ENQ	One or more blanks must follow ENQ.
b̸	
(	
qname addr	*qname addr:* A-type address, or register (2) - (12).
,*rname addr*	*rname addr:* A-type address, or register (2) - (12).
, ,E ,S	**Default:** E
, ,*rname length*	*rname length:* symbol, decimal digit, or register (2) - (12). **Default:** assembled length of *rname* **Note:** *rname length* must be coded if a register is specified for *rname addr.*
, ,STEP ,SYSTEM ,SYSTEMS	**Default:** STEP
,*var12345*	*var12345:* The preceding 5 parameters may be repeated up to 65,535 times.
)	
,RET=CHNG ,RET=HAVE ,RET=TEST ,RET=USE ,RET=NONE	**Default:** RET=NONE
,SMC=NONE ,SMC=STEP ,ECB=*ecb addr* ,TCB=*tcb addr*	*ecb addr:* A-type address, or register (2) - (12). *tcb addr:* A-type address, or register (2) - (12). **Default:** SMC=NONE **Note:** ECB cannot be specified with RET above. **Note:** TCB cannot be specified with RET=HAVE or RET=NONE above.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters restricted in use are explained below. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

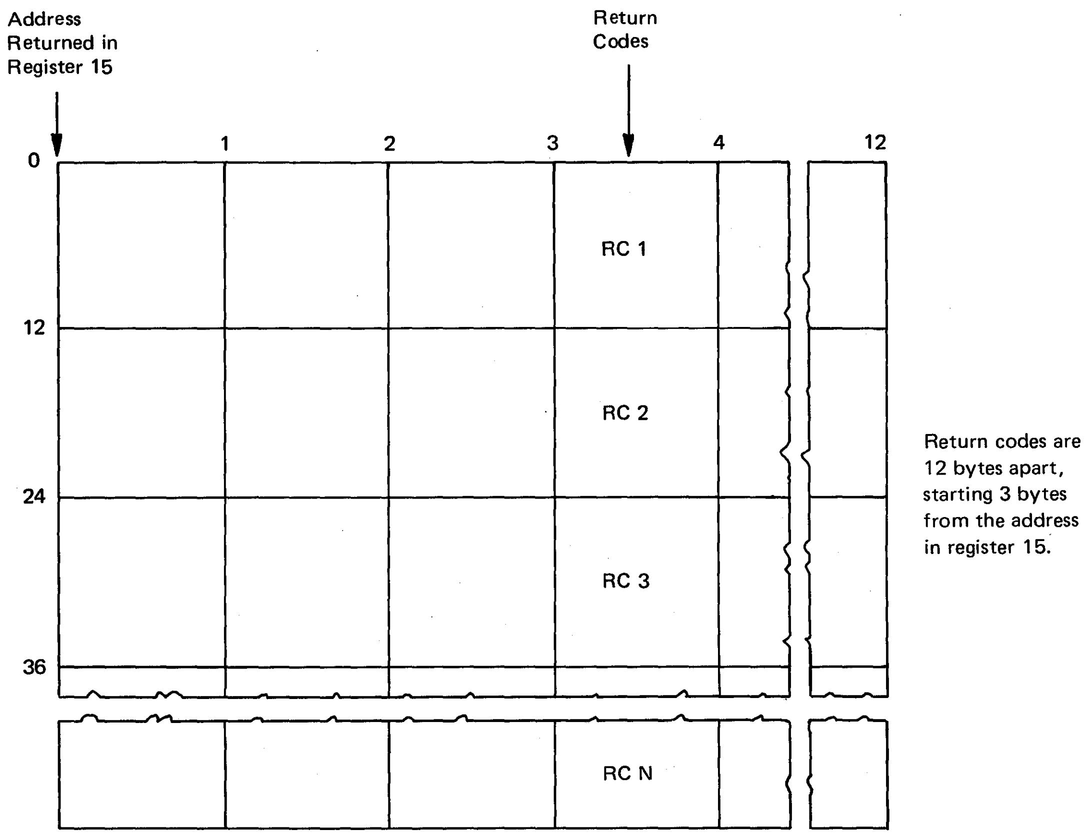
,SMC = NONE
,SMC = STEP
,ECB = *ecb addr*
,TCB = *tcb addr*
specifies optional parameters available to the system programmer:

SMC specifies that the set-must-complete function is not to be used (NONE) or that it is to place other tasks for the step nondispatchable until the requesting task has completed its operations on the resource (STEP).

ECB specifies the address of an ECB, and conditionally requests all of the resources named in the macro instruction.

TCB specifies the address of a fullword on a fullword boundary that contains the address of a TCB on whose behalf the ENQ is to be done.

Return codes are provided by the control program only if you specify RET=TEST, RET=USE, RET=CHNG, or RET=HAVE; otherwise return of the task to the active condition indicates that control of the resource has been assigned to the task. If all return codes for the resources named in the ENQ macro instruction are 0, register 15 contains 0. If any of the return codes are not 0, register 15 contains the address of a storage area containing the return codes, as shown in Figure 28.



Address Returned in Register 15
Return Codes
0
1
2
3
4
12
12
24
36
RC 1
RC 2
RC 3
RC N
Return codes are 12 bytes apart, starting 3 bytes from the address in register 15.

Figure 28. Return Code Area Used by ENQ

The return codes are placed in the parameter list resulting from the macro expansion in the same sequence as the resource names in the ENQ macro instruction. The return codes are shown below.

Hexadecimal Code	Meaning
0	For RET=TEST, the resource was immediately available. For RET=USE, RET=HAVE, or ECB=, control of the resource has been assigned to the active task. For RET=CHNG, the status of the resource has been changed to exclusive.
4	For RET=TEST or RET=USE, the resource is not immediately available. For RET=CHNG, the status cannot be changed to shared. For ECB=, the ECB will be posted when available.
8	For RET=TEST, RET=USE, RET=HAVE, or ECB=, a previous request for control of the same resource has been made for the same task. Task has control of resource. For RET=CHNG, the resource has not been queued. If bit 3 is on -- shared control of resource; if bit 3 is off -- exclusive control.
20	A previous request for control of the same resource has been made for the same task. Task does not have control of resource.

Example 1

Operation: Unconditionally request exclusive control of a serially reusable resource that is known only within the address space (STEP), and place other tasks for the step nondispatchable until the requesting task has completed its operations on the resource.

```
ENQ      (MAJOR1,MINOR1,E,8,STEP),SMC=STEP
```

Example 2

Operation: Conditionally request control of a serially reusable resource in behalf of another task. The resource is known by more than one address space, and is only wanted if immediately available.

```
ENQ      (MAJOR2,MINOR2,S,4,SYSTEM),TCB=(R2),RET=USE
```

ENQ (List Form)

The list form of the ENQ macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ENQ.
ENQ	
ƀ	One or more blanks must follow ENQ.
(	
qname addr	*qname addr:* A-type address.
, ,*rname addr*	*rname addr:* A-type address.
, ,E ,S	**Default:** E
, ,*rname length*	*rname length:* symbol or decimal digit. **Default:** assembled length of *rname*
, ,STEP ,SYSTEM ,SYSTEMS	**Default:** STEP
,*var12345*	*var12345:* The preceding 5 parameters may be repeated up to 65,535 times.
)	
,RET=CHNG ,RET=HAVE ,RET=TEST ,RET=USE ,RET=NONE	**Default;** RET=NONE
,SMC=NONE ,SMC=STEP ,ECB=*ecb addr* ,TCB=0	*ecb addr:* A-type address. **Default:** SMC=NONE **Note:** ECB cannot be specified with RET above. **Note:** TCB cannot be specified with RET=HAVE or RET=NONE above, and must be specified on the list form if used on the execute form.
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

The parameters restricted in use are explained under the standard form of the ENQ macro instruction. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

ENQ (Execute Form)

The execute form of the ENQ macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ENQ.
ENQ	
ƀ	One or more blanks must follow ENQ.
(	**Note:** (and) are the beginning and end of a parameter list. The entire list is optional. If nothing in the list is desired then (,), and all parameters between (and) should not be specified. If something in the list is desired, then (,), and all parameters in the list should be specified as indicated at the left.
qname addr	*qname addr:* RX-type address, or register (2) - (12).
, ,*rname addr*	*rname addr:* RX-type address, or register (2) - (12).
, ,E ,S	
, ,*rname length*	*rname length:* symbol, decimal digit, or register (2) - (12).
, ,STEP ,SYSTEM ,SYSTEMS	
,*var12345*	*var12345:* The preceding 5 parameters may be repeated up to 65,535 times.
)	**Note:** See note opposite (above.
,RET=CHNG ,RET=HAVE ,RET=TEST ,RET=USE ,RET=NONE	
,SMC=NONE ,SMC=STEP ,ECB=*ecb addr* ,TCB=*tcb addr*	*ecb addr:* A-type address, or register (2) - (12). *tcb addr:* A-type address, or register (2) - (12). **Note:** ECB cannot be specified with RET above. **Note:** TCB cannot be specified with RET=HAVE or RET=NONE above.
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E ,*ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

The parameters restricted in use are explained under the standard form of the ENQ macro instruction. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

Note: If ECB (or TCB) is specified in the execute form, ECB=0 (or TCB=0) must be specified in the list form.

ESTAE — Extended STAE

The ESTAE macro instruction is described in the **OS/VS2 Supervisor Services and Macro Instructions**, with the exception of the RECORD, BRANCH, and SVEAREA parameters. These parameters are restricted in use and should only be used by types 2, 3, and 4 SVCs executing in supervisor state, under protection key 0, and owning the local lock.

The syntax of the complete ESTAE macro instruction is shown below. However, only the explanation of the restricted parameters is presented. Explanation of the other parameters can be found in **OS/VS2 Supervisor Services and Macro Instructions.**

The standard form of the ESTAE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ESTAE.
ESTAE	
ƀ	One or more blanks must follow ESTAE.
exit addr 0	*exit addr:* A-type address, or register (2) - (12).
,CT ,OV	**Default:** CT
,PARAM=*list addr*	*list addr:* A-type address, or register (2) - (12).
,XCTL=NO ,XCTL=YES	**Default:** XCTL=NO
,PURGE=NONE ,PURGE=QUIESCE ,PURGE=HALT	**Default:** PURGE=NONE
,ASYNCH=YES ,ASYNCH=NO	**Default:** ASYNCH=YES
,TERM=NO ,TERM=YES	**Default:** TERM=NO
,BRANCH=NO ,BRANCH=YES, SVEAREA=*save area*	**Default:** BRANCH=NO *save addr:* A-type address, or register (2) - (12) or (13).
,RECORD=NO ,RECORD=YES	**Default:** RECORD=NO
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters restricted in use are explained below. The explanation of the other parameters is as explained in **OS/VS2 Supervisor Services and Macro Instructions.**

,BRANCH = NO
,BRANCH = YES,SVEAREA = *save addr*

specifies that an SVC 60 entry to the ESTAE service routine is to be performed (NO) or that a branch entry is to be performed (YES). (The branch entry is for type 2, 3, or 4 SVCs only.) The save area is a 72-byte area used to save the general registers. If BRANCH=YES is specified, the caller must be in key 0 and own the local lock.

,RECORD = NO
,RECORD = YES

specifies that the SDWA workarea will not be written to SYS1.LOGREC (NO) or that the entire SDWA workarea, both fixed and variable, will be written to SYS1.LOGREC (YES).

Control is returned to the instruction following the ESTAE macro instruction. When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful completion of ESTAE request.
04	ESTAE OV was specified with a valid exit address, but the current exit is either nonexistent, not owned by the user's RB, or is not an ESTAE exit.
08	BRANCH=YES was issued for the current SVRB with a create request; the previous BRANCH=YES exit is canceled and the new exit is made the current exit.
0C	Cancel or an exit address equal to zero was specified, and either there are no exits for this TCB, the most recent exit is not owned by the caller, or the most recent exit is not an ESTAE exit.
10	An unexpected error was encountered while processing this request.
14	ESTAE was unable to obtain storage for an SCB.

Example 1

Operation: Take the ESTAE exit specified by register 4, allow asynchronous exit processing, do not allow special error processing, do not branch enter SVC 60, and default to CT (create) and PURGE=NONE.

```
ESTAE    (4),ASYNCH=YES,TERM=NO,BRANCH=NO
```

ESTAE (List Form)

The list form of the ESTAE macro instruction is used to construct a remote control program parameter list.

The list form of the ESTAE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ESTAE.
ESTAE	
ƀ	One or more blanks must follow ESTAE.
exit addr	*exit addr:* A-type address.
,PARAM=*list addr*	*list addr:* A-type address.
,PURGE=NONE ,PURGE=QUIESCE ,PURGE=HALT	**Default:** PURGE=NONE
,ASYNCH=YES ,ASYNCH=NO	**Default:** ASYNCH=YES
,TERM=NO ,TERM=YES	**Default:** TERM=NO
,RECORD=NO ,RECORD=YES	**Default:** RECORD=NO
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

The parameters restricted in use are explained under the standard form of the ESTAE macro instruction. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

ESTAE (Execute Form)

A remote control program parameter list is used in, and can be modified by, the execute form of the ESTAE macro instruction. The control program parameter list can be generated by the list form of the ESTAE macro instruction. If the user desires to dynamically changed the contents of the remote ESTAE parameter list, he may do so by coding a new exit address and/or a new parameter list address. If exit address or PARM= is coded, only the associated field in the remote ESTAE parameter list will be changed. The other field will remain as it was before the current ESTAE request was made.

The execute form of the ESTAE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ESTAE.
ESTAE	
ƀ	One or more blanks must follow ESTAE.
exit addr 0	*exit addr:* RX-type address, or register (2) - (12).
,CT ,OV	*list addr:* RX-type address, or register (2) - (12).
,PARAM=*list addr*	
,XTCL=NO ,XCTL=YES	
,PURGE=NONE ,PURGE=QUIESCE ,PURGE=HALT	
,ASYNCH=YES ,ASYNCH=NO	
,TERM=NO ,TERM=YES	
,BRANCH=NO ,BRANCH=YES, SVEAREA=*save addr*	*save addr:* RX-type address, or register (2) - (12) or (13).
,RECORD=NO ,RECORD=YES	
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E ,*ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

The parameters restricted in use are explained under the standard form of the ESTAE macro instruction. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

Example 1

Operation: Take the ESTAE exit labeled ADDR, allow synchronous exit processing, halt I/O, allow special error processing, branch enter SVC 60, provide 72-byte save area at SADDR, and execute the execute form of the macro instruction. EXEC is the label of the ESTAE parameter list built by the list form of the macro instruction.

```
ESTAE    ADDR,ASYNCH=YES,PURGE=HALT,TERM=YES,BRANCH=YES,
         SVEAREA=SADDR,MF=(E,EXEC)
```

EVENTS — Wait for Events

The EVENTS macro instruction is described in the **OS/VS2 Supervisor Services and Macro Instructions**, with the exception of the BRANCH parameter. This parameter is restricted in use and should only be used by programs executing in supervisor state, under protection key 0, and owning the local lock.

The syntax of the complete EVENTS macro instruction is shown below. However, only the explanation of the restricted parameter is presented. Explanation of the other parameters can be found in **OS/VS2 Supervisor Services and Macro Instructions.**

The EVENTS macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede EVENTS.
EVENTS	
ƀ	One or more blanks must follow EVENTS.

ENTRIES=*nmbr*
ENTRIES=DEL,TABLE=*tab addr*
TABLE=*tab addr*
	nmbr: decimal digits 1-32767.
	tab addr: symbol, RX-type address, or register (2) -(12).
	Note: If the ENTRIES parameter is specified as indicated in the first two formats, no other parameters may be specified.

,ECB=*ecb addr* ,LAST=*last addr*	*ecb addr*: symbol, RX-type address, or register (2) - (12). *last addr*: symbol, RX-type address, or register (2) - (12). **Note**: If LAST is specified, WAIT must also be specified.
,WAIT=YES ,WAIT=NO	
,BRANCH=NO ,BRANCH=YES	**Default**: BRANCH=NO

The parameters restricted in use are explained below. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

,BRANCH = NO
,BRANCH = YES
specifies that an SVC entry (BRANCH=NO) or a branch entry (BRANCH=YES) is to be performed.

EXTRACT — Extract TCB Information

The EXTRACT macro instruction causes the control program to provide information from specified fields of the task control block or a subsidiary control block for either the active task or one of its subtasks. The information is placed in an area provided by the problem program.

The standard form of the EXTRACT macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b̸	One or more blanks must precede EXTRACT.
EXTRACT	
b̸	One or more blanks must follow EXTRACT.
answer addr	*answer addr:* A-type address, or register (2) - (12).
,'S' *,tcb addr*	*tcb addr:* A-type address, or register (2) - (12). **Default:** 'S'
,FIELDS=*(tcb info)*	*tcb info:* any combination of the following, separated by commas: ALL PRI TSO GRS CMC PSB FRS TIOT TJID AETX COMM ASID

The parameters are explained below:

answer addr
specifies the address of the answer area to contain the requested information. The address is of one or more fullwords, starting on a fullword boundary. The number of fullwords required is the same as the number of fields specified in the FIELDS parameter, unless ALL is coded. If ALL is coded, seven fullwords are required.

,'S'
,tcb addr
specifies the address of a fullword on a fullword boundary containing the address of a task control block for a subtask of the active task. If 'S' is coded or assumed, no address is specified and the active task is assumed.

,FIELDS=*(tcb info)*
specifies the task control block information requested:

ALL requests information from the GRS, FRS, reserved, AETX, PRI, CMC, and TIOT fields. (If ALL is specified, 7 words are required just for ALL.)

GRS is the address of the save area used by the control program to save the general registers 0-15 when the task is not active.

FRS is the address of the save area used by the control program to save the floating point registers 0, 2, 4, and 6 when the task is not active.

AETX is the address of the end of task exit routine specified in the ETXR parameter of the ATTACH macro instruction used to create the task.

PRI is the current limit (third byte) and dispatching (fourth byte) priorities of the task. The two high-order bytes are set to zero.

CMC is the task completion code. If the task is not complete, the field is set to zero.

TIOT is the address of the task input/output table.

COMM is the address of the command scheduler communications list. The list consists of a pointer to the communications event control block and a pointer to the command input buffer. The high-order bit of the last pointer is set to one to indicate the end of the list.

TSO is the address of a byte in which a high bit of 1 indicates a TSO address space, and a high bit of 0 indicates a non-TSO address space.

PSB is the address of the protected storage control block, which is extracted from the job step control block.

TJID is the address space identifier (ASID) for a TSO address space, and zero for a non-TSO address space.

ASID is the address space identifier.

Example 1

Operation: Provide information from all the fields of the indicated TCB except ASID. WHERE is the label of the answer area, ADDRESS is the label of a fullword which contains the address of the subtask TCB for which information is to be extracted.

```
EXTRACT WHERE,ADDRESS,FIELDS=(ALL,TSO,COMM,PSB,TJID)
```

Example 2

Operation: Provide information from the current TCB, as above.

```
EXTRACT WHERE,'S',FIELDS=(ALL,TSO,COMM,PSB,TJID)
```

EXTRACT (List Form)

The list form of the EXTRACT macro instruction is used to construct a remote control program parameter list.

The list form of the EXTRACT macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede EXTRACT.
EXTRACT	
ƀ	One or more blanks must follow EXTRACT.
answer addr	*answer addr:* A-type address.
,'S' *,tcb addr*	*tcb addr:* A-type address. *Default:* 'S'
,FIELDS=*(tcb info)*	*tcb info:* any combination of the following, separated by commas: ALL PRI TSO GRS CMC PSB FRS TIOT TJID AETX COMM ASID
,MF=L	

The parameters are explained under the standard form of the EXTRACT macro instruction, with the following exceptions:

,MF=L
specifies the list form of the EXTRACT macro instruction.

EXTRACT (Execute Form)

The execute form of the EXTRACT macro instruction uses, and can modify, a remote control program parameter list. If the FIELDS parameter restricted in use is coded in the execute form, any TCB information specified in a previous FIELDS parameter is cancelled and must be respecified if required for this execution of the macro instruction.

The execute form of the EXTRACT macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede EXTRACT.
EXTRACT	
ƀ	One or more blanks must follow EXTRACT.
answer addr	*answer addr:* RX-type address, or register (2) - (12).
,'S' *,tcb addr*	*tcb addr:* RX-type address, or register (2) - (12).
,FIELDS=*(tcb info)*	*tcb info:* any combination of the following, separated by commas: ALL PRI TSO GRS CMC PSB FRS TIOD TJID AETX COMM ASID
,MF=(E, *ctrl addr)*	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

The parameters are explained under the standard form of the EXTRACT macro instruction, with the following exceptions:

,MF=(E, *ctrl addr)*
specifies the execute form of the EXTRACT macro instruction using a remote control program parameter list.

FREEMAIN — Free Virtual Storage

The FREEMAIN macro instruction is described in the **OS/VS2 Supervisor Services and Macro Instructions**, with the exception of the BRANCH and KEY parameters. These parameters are restricted in use and should only be used by programs executing in supervisor state, under protection key 0.

Note: This macro instruction requires that the CVT mapping macro be assembled into the caller's csect.

The syntax of the complete FREEMAIN macro instruction is shown below. However, only the explanation of the restricted parameters is presented. Explanation of the other parameters can be found in **OS/VS2 Supervisor Services and Macro Instructions.**

The standard form of the FREEMAIN macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b̸	One or more blanks must precede FREEMAIN.
FREEMAIN	
b̸	One or more blanks must follow FREEMAIN.
LC,LA=*length addr* LU,LA=*length addr* L,LA=*length addr* VC VU V EC,LV=*length value* EU,LV=*length value* E,LV=*length value* RC,LV=*length value* RC,SP=*subpool nmbr* RU,LV=*length value* RU,SP=*subpool nmbr* R,LV=*length value* R,SP=*subpool nmbr*	*length addr:* A-type address, or register (2) - (12). *length value:* symbol, decimal digit, or register (2) - (12). If R is specified, register (0) may also be specified. *subpool nmbr:* symbol, decimal digit 0-127, or register (0) or (2) - (12). If R,SP=(0) is specified, the high order byte of register 0 must contain the subpool number and the low order 3 bytes must contain the length value. **Note:** For a subpool FREEMAIN, if the formats RC,SP=*subpool nmbr* or RU,SP=*subpool nmbr* or R,SP=*subpool nmbr* are specified, no other parameters may be specified.
,A=*addr*	*addr:* A-type address, or register (2) - (12).
,SP=*subpool nmbr*	*subpool nmbr:* symbol, decimal digit 0-127, or register (0) or (2) - (12). If R,SP=(0) is specified, the high order byte of register 0 must contain the subpool number and the low order 3 bytes must contain the length value.
,BRANCH=YES ,BRANCH=(YES,GLOBAL)	**Note:** BRANCH=(YES,GLOBAL) may only be specified with RC or RU above.
,KEY=*nmbr*	*nmbr:* decimal digits 0-15, or register (2) - (12). **Note:** This parameter may be specified only if BRANCH above is also specified.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters restricted in use are explained below. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instruction.**

,BRANCH = YES
,BRANCH = (YES,GLOBAL)
specifies that a branch entry is to be used instead of an SVC entry. If (YES,GLOBAL) is specified, the GLBRANCH entry point to service global storage requests without the need for the local memory lock will be used; the SALLOC lock must be held.

If BRANCH=YES is specified, the caller must pre-load register 4 with the TCB address, pre-load register 7 with the ASCB address, and hold the local address space lock prior to entry. Register 7 will not contain the ASCB address when control is returned to the caller. Register 3 will also be destroyed if RC or RU was specified.

If BRANCH=(YES,GLOBAL) is specified, registers 4 and 7 need not contain the TCB and ASCB addresses; and registers 3 and 4 will be changed when control is returned to the caller. Additionally, the SP parameter may only designate subpools 227, 228, 231, 239, 241, or 245.

The FREEMAIN macro instruction, with BRANCH=(YES,GLOBAL) specified, requires that the IHAWSAVT mapping macro be assembled into the caller's csect.

,KEY=*key nmbr*

specifies the key (in bits 24-27 of the register) in which the requested storage was obtained. This parameter applies to subpools 227, 228, 229, 230, 231, and 241, and allows both global and local storage to be freed in the requestor's storage protection key.

When control is returned, register 15 may contain the following return code:

Hexadecimal Code	**Meaning**
8	Part of area being freed is still fixed.

The parameters restricted in use are explained under the standard form of the FREEMAIN macro instruction. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

Example 1

Operation: Free 400 bytes of storage addressed by register 1 via a branch entry. If the storage is successfully freed, register 15 will contain 0; otherwise, register 15 will contain a nonzero value.

```
FREEMAIN    EC,LV=400,A=(1),BRANCH=YES
```

Example 2

Operation: Free all storage in subpool 239. Register 3 has been preset to contain the storage key of the storage to be released. If the request is unsuccessful, the caller will be abnormally terminated.

```
FREEMAIN    RU,SP=239,KEY=(3),BRANCH=(YES,GLOBAL)
```

FREEMAIN (List Form)

The list form of the FREEMAIN macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b̶	One or more blanks must precede FREEMAIN.
FREEMAIN	
b̶	One or more blanks must follow FREEMAIN.
LC LU L VC VU V EC EU E	
,LA=*length addr* ,LV=*length value*	*length addr:* A-type address. *length value:* symbol or decimal digit. **Note:** LA may only be specified with LC, LU, or L above. **Note:** LV may only be specified with EC, EU, or E above.
,A=*addr*	*addr:* A-type address.
,SP=*subpool nmbr*	*subpool nmbr:* symbol or decimal digit 0 - 127.
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

The parameters restricted in use are explained under the standard form of the FREEMAIN macro instruction. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

FREEMAIN (Execute Form)

The execute form of the FREEMAIN macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede FREEMAIN.
FREEMAIN	
ƀ	One or more blanks must follow FREEMAIN.
LC LU L VC VU V EC EU E	
,LA=*length addr* ,LV=*length value*	*length addr:* RX-type address or register (2) - (12). *length value:* symbol, decimal digit, or register (2) - (12). **Note:** LA may only be specified with LC, LU, or L above. **Note:** LV may only be specified with EC, EU, or E above.
,A=*addr*	*addr:* RX-type address, or register (2) - (12)
,SP=*subpool nmbr*	*subpool nmbr:* symbol, decimal digit, or register (0) or (2) - (12).
,BRANCH=YES	
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E ,*ctrl prog*)	*ctrl prog:* RX-type address, or register (1) or (2) - (12).

GETMAIN — Allocate Virtual Storage

The GETMAIN macro instruction is described in the **OS/VS2 Supervisor Services and Macro Instructions**, with the exception of the BRANCH and KEY parameters. These parameters are restricted in use and should only be used by programs executing in supervisor state, under protection key 0.

Note: This macro instruction requires that the CVT mapping macro instruction be assembled into the caller's csect.

The syntax of the complete GETMAIN macro instruction is shown below. However, only the explanation of the restricted parameters is presented. Explanation of the other parameters can be found in **OS/VS2 Supervisor Services and Macro Instructions.**

The standard form of the GETMAIN macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GETMAIN.
GETMAIN	
ƀ	One or more blanks must follow GETMAIN.
LC,LA=*length addr*,A=*addr* LU,LA=*length addr*,A=*addr* VC,LA=*length addr*,A=*addr* VU,LA=*length addr*,A=*addr* EC,LV=*length value*,A=*addr* EU,LV=*length value*,A=*addr* RC,LV=*length value* RU,LV=*length value* R,LV=*length value*	*length addr:* A-type address, or register (2) - (12). *length value:* symbol, decimal digit, or register (2) - (12). If R is specified, register (0) may also be specified. *addr:* A-type address, or register (2) - (12).
,SP=*subpool nmbr*	*subpool nmbr:* symbol, decimal digit 0 - 127, or register (0) or (2) - (12). **Note:** If R,LV=(0) is specified above, SP may not be specified.
,BNDRY=DBLWD ,BNDRY=PAGE	**Default:** BNDRY=DBLWD **Note:** This parameter may not be specified with R above.
,BRANCH=YES ,BRANCH=(YES,GLOBAL)	**Note:** BRANCH=(YES,GLOBAL) may only be specified with RC or RU above.
,KEY=*key number*	*key nmbr:* decimal digits 0-15, or register (2) - (12). **Note:** This parameter may be specified only if BRANCH above is also specified.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters restricted in use are explained below. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

,BRANCH = YES
,BRANCH = (YES,GLOBAL)

specifies that a branch entry is to be used instead of an SVC entry. If (YES,GLOBAL) is specified, the GLBRANCH entry point to service global storage requests without the need for the local memory lock will be used; the SALLOC lock must be held.

If BRANCH=YES is specified, the caller must pre-load register 4 with the TCB address, pre-load register 7 with the ASCB address, and hold the local address space lock prior to entry. Register 7 will not contain the ASCB address when control is returned to the caller. Register 3 will also be destroyed if RC or RU was specified.

If BRANCH=(YES,GLOBAL) is specified, registers 4 and 7 need not contain the TCB and ASCB addresses; and register 3 and 4 will be changed when control is returned to the caller. Additionally, the SP parameter may only designate subpools 227, 228, 231, 239, 241, or 245.

The GETMAIN macro instruction, with this parameter specified, requires that the IHAWSAVT mapping macro be assembled into the caller's csect.

,KEY=*key nmbr*
specifies the key (in bits 24-27 of the register) in which the requested storage is to be obtained. This parameter applies to subpools 227, 228, 229, 230, 231, and 241, and allows both global and local storage to be obtained in the requester's storage protection key.

When control is returned, register 15 may contain the following return code:

Hexadecimal Code	**Meaning**
8	On request for SQA or LSQA, no real storage page is available.

Example 1

Operation: Obtain 248 bytes of storage from the user's region via a branch entry. If the routine is in supervisor state, subpool 252 will be used; otherwise, subpool 0 will be used. If the storage cannot be obtained, the caller will be abnormally terminated.

```
GETMAIN EU,LV=248,A=AREAADDR,BRANCH=YES
```

Example 2

Operation: Obtain one page of storage from the common service area, and cause the acquired storage to be initialized with a storage key of 9. A return code of 0 (if successful) or 4 (if unsuccessful) will be returned.

```
GETMAIN RC,LV=4096,SP=231,BRANCH=(YES,GLOBAL),BNDRY=PAGE,KEY=9
```

GETMAIN (List Form)

The list form of the GETMAIN macro instruction is written as follows:

name	*name:* Begin *name* in column 1.
ƀ	One or more blanks must precede GETMAIN.
GETMAIN	
ƀ	One or more blanks must follow GETMAIN.
LC LU VC VU EC EU	
,LA=*length addr* ,LV=*length value*	*length addr:* A-type address. *length value:* symbol or decimal digit. **Note:** LA may be specified with EC or EU above. **Note:** LV may not be specified with LC, LU, VC or VU above.
,A=*addr*	*addr:* A-type address.
,SP=*subpool nmbr*	*subpool nmbr:* symbol or decimal digit 0-127.
,BNDRY=DBLWD ,BNDRY=PAGE	**Default:** BNDRY=DBLWD
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

The parameters restricted in use are explained under the standard form of the GETMAIN macro instruction. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

GETMAIN (Execute Form)

The execute form of the GETMAIN macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GETMAIN.
GETMAIN	
ƀ	One or more blanks must follow GETMAIN.
LC LU VC VU EC EU	
,LA=*length addr* ,LV=*length value*	*length addr:* RX-type address or register (2) - (12). *length value:* symbol, decimal digit, or register (2) - (12). **Note:** LA may not be specified with EC or EU above. **Note:** LV may not be specified with LC, LU, VC, or VU above.
,A=*addr*	*addr:* RX-type address, or register (2) - (12).
,SP=*subpool nmbr*	*subpool nmbr:* symbol, decimal digit 0-127, or register (0) or (2) - (12).
,BNDRY=DBLWD ,BNDRY=PAGE	**Default:** BNDRY=DBLWD
,BRANCH=YES	
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E ,*ctrl prog*)	*ctrl prog:* RX-type address, or register (1) or (2) - (12).

The parameters restricted in use are explained under the standard form of the GETMAIN macro instruction. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

MODESET — Change System Status

The MODESET macro instruction is used to change system status by altering the PSW key or mode indicator. It causes a supervisor routine (IEAVMODE) to alter the RB old program status word (RBOPSW) so that the desired PSW will be loaded when MODESET returns to the caller. MODESET also generates inline code that saves and/or changes the protection key in the current PSW.

The standard form of the MODESET macro instruction has two forms: the form that generates an SVC and the form that generates inline code. The form that generates inline code uses the SPKA instruction (see **IBM System/370 Principles of Operation)**, and is executable only in supervisor state. The form that generates an SVC is executable by users in supervisor state, under protection key 0-7, or APF-authorized.

The standard form of the MODESET macro instruction that generates inline code is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede MODESET.
MODESET	
ƀ	One or more blanks must follow MODESET.

EXTKEY=*key*
KEYADDR=*key addr*
	key: one of the following:
	SCHED SRM ZERO
	JES SUPR TCB
	RSM DATAMGT RBT1
	VSM TCAM RBT234
	key addr: A-type address or register (2).
	Note: The WORKREG parameter is required if the following are specified:
	EXTKEY=TCB EXTKEY=RBT234
	EXTKEY=RBT1 KEYADDR=A-type address

,SAVEKEY=*old key addr*	*old key addr:* A-type address or register (2). **Note:** If KEYADDR=(2) is specified above, then SAVEKEY=(2) cannot be specified. **Note:** The WORKREG parameter is required if SAVEKEY=A-type address is specified.
,WORKREG=*reg*	*reg:* decimal digits 0 - 15.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained below:

EXTKEY=*key*
KEYADDR=*key addr*
specifies the key to be set in the current PSW or the address of the key.

SCHED — Scheduler key.

JES — Job entry subsystem key.

RSM — Real storage management key.

VSM — Virtual storage management key.

SRM — System resource management key.

SUPR — Supervisor key.

DATAMGT — Data management key.

TCAM — Telecommunications access method key.

ZERO — Key of zero is to be set.

TCB — Key is to be obtained from TCB field TCBPKF.

RBT1 — Key is to be obtained from the RBOPSW field of the active RB of type 1 SVC routine issuing MODESET.

RBT234 — Key is to be obtained from the RBOPSW field of the RB preceding SVRB of type 2, 3, or 4 SVC routine issuing MODESET.

KEYADDR specifies a location 1 byte in length which contains the key in bit positions 0-3. If register (2) is specified, the key is contained in bit positions 24-27 (bits 28-31 are ignored). This parameter permits a previously saved key to be restored.

,SAVEKEY=*old key addr*
specifies a location 1 byte in length where the current PSW key is to be saved, in bit positions 0-3. If register (2) is specified, the key is left in register 2.

,WORKREG=*reg*
specifies the register into which the contents of register 2 are to be saved while performing the SAVEKEY function, or the working register to be used by the EXTKEY or KEYADDR function. If WORKREG=2 is specified, no register saving takes place.

,RELATED=*value*
specifies information used to self-document macro instructions by 'relating' functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

The RELATED parameter is available on macro instructions that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE, and on macro instructions that relate to previous occurrences of the same macro instructions (for example, CHAP and ESTAE).

The parameter may be used, for example, as follows:

```
GET1     GETMAIN     R,LV=4096,RELATED=(FREE1,'GET STORAGE')
FREE1    FREEMAIN    R,LV=4096,A=(1),RELATED=(GET1,'FREE STORAGE')
```

Note: This form of the macro instruction does not generate any return codes.

The standard form of the MODESET macro instruction that generates an SVC is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b̸	One or more blanks must precede MODESET.
MODESET	
b̸	One or more blanks must follow MODESET.

KEY=ZERO
KEY=NZERO
	Note: KEY is required only if no other parameter is specified.

,MODE=PROB ,MODE=SUP	**Note:** MODE is required only if no other parameter is specified.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained below:

KEY = ZERO
KEY = NZERO
specifies that the PSW key (bits 8-11) is to be either set to zero (ZERO) or set to the value in the caller's TCB (NZERO).

,MODE = PROB
,MODE = SUP
specifies that the PSW mode indicator (bit 15) is to be either turned on (PROB) or turned off (SUP).

Note: This form of the macro instruction does not generate any return codes.

Example 1

Operation: Save the current PSW key, and change the key to that of the scheduler.

```
MODESET EXTKEY=SCHED,SAVEKEY=KEYSAVE,WORKREG=1
```

Example 2

Operation: Change to supervisor mode and key zero.

```
MODESET KEY=ZERO,MODE=SUP
```

MODESET (List Form)

The list form of the MODESET macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede MODESET..
MODESET	
ƀ	One or more blanks must follow MODESET.
RELATED=*value*,	*value:* any valid macro keyword specification.
MF=L	

The parameters are explained under the standard form of the MODESET macro instruction, with the following exceptions:

MF = L
specifies the list form of the MODESET macro instruction.

MODESET (Execute Form)

The execute form of the MODESET macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede MODESET.
MODESET	
ƀ	One or more blanks must follow MODESET.
RELATED=*value*,	*value:* any valid macro keyword specification.
MF=(E ,*list addr*)	*list addr:* RX-type address, or register (1).

The parameters are explained under the standard form of the MODESET macro instruction, with the following exceptions:

MF=(E ,*list addr*)
specifies the execute form of the MODESET macro instruction, using a parameter list address.

Incompatible Parameters

The ENABLE and SYSMASK parameters, available with Release 1 of VS2, are no longer supported on MODESET. The functions formerly available via ENABLE and SYSMASK are now provided by the SETLOCK macro instruction and the STNSM and STOSM instructions.

NIL — Provide a Lock Via an AND IMMEDIATE (NI) Instruction

The NIL macro instruction is used to provide a lock on a byte of storage on which an AND IMMEDIATE (NI) instruction is to be executed. The byte of storage exists in a multiprocessing environment and the possibility exists that the byte may be changed by another CPU at the same time. Storage modification during NIL processing is accomplished in the same manner as that used by the COMPARE AND SWAP (CS) instruction

For details on the AND IMMEDIATE and COMPARE AND SWAP instructions, see **IBM System/370 Principles of Operation.**

The NIL macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks msut precede NIL.
NIL	
ƀ	One or more blanks must follow NIL.
byte addr	*byte addr:* A-type address.
,mask	*mask:* symbol or self defining term.
,REF=stor addr	*stor addr:* A-type address.
,WREGS=*(reg1,reg2,reg3)* ,WREGS=*(reg1,reg2)* ,WREGS=*(reg1,,reg3)* ,WREGS=*(,reg2,reg3)* ,WREGS=*(reg1)* ,WREGS=*(,reg2)* ,WREGS=*(,,reg3)*	*reg1:* symbol, or decimal digits 0-16. *reg2:* symbol, or decimal digits 1-16. *reg3:* symbol, or decimal digits 0-16. **Default** for *reg1:* 0 **Default** for *reg2:* 1 **Default** for *reg3:* 2

The parameters are explained below:

byte addr
specifies the address of the byte to which the AND function is to be applied.

,mask
specifies the value to be ANDed to the byte at the address specified above.

,REF=stor addr
specifies the address of a storage location on a fullword boundary. This address provides the means by which the COMPARE AND SWAP instruction may be executed. The address must be less than or equal to the byte address specified above, and the difference between the addresses must be less than 4096. The two addresses must be addressable via the same base register.

,WREGS = *(reg1,reg2,reg3)*
,WREGS = *(reg1,reg2)*
,WREGS = *(reg1,,reg3)*
,WREGS = *(,reg2,reg3)*
,WREGS = *(reg1)*
,WREGS = *(,reg2)*
,WREGS = *(,,reg3)*
specifies the work registers to be used to perform the COMPARE AND SWAP instruction. *reg1* is used to contain the 'old' byte; *reg2* is used to contain the 'updated' byte; and *reg3* is used to contain the mask.

Example 1

Operation: Provide a lock on the byte of storage specified by the address STRESTAT. UCBOB is the address used to reference the byte, and FSRTECGS is the mask used.

```
NIL SRTESTAT,FSRTECGS,WREGS=(15,4,5),REF=UCBOB
```

OIL — Provide a Lock Via an OR IMMEDIATE (OI) Instruction

The OIL macro instruction is used to provide a lock on a byte of storage on which an OR IMMEDIATE (OI) instruction is to be executed. The byte of storage exists in a multiprocessing environment and the possibility exists that the byte may be changed by another CPU at the same time. Storage modification during OIL processing is accomplished in the same manner as that used by the COMPARE AND SWAP (CS) instruction.

| For details on the OR IMMEDIATE and COMPARE AND SWAP instructions, see **IBM System/370 Principles of Operation.**

The OIL macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede OIL.
OIL	
ƀ	One or more blanks must follow OIL.
byte addr	*byte addr:* A-type address.
,mask	*mask:* symbol or self defining term.
,REF=stor addr	*stor addr:* A-type address.
,WREGS=*(reg1,reg2,reg3)* ,WREGS=*(reg1,reg2)* ,WREGS=*(reg1,,reg3)* ,WREGS=*(,reg2,reg3)* ,WREGS=*(reg1)* ,WREGS=*(,reg2)* ,WREGS=*(,,reg3)*	*reg1:* symbol, or decimal digits 0-16. *reg2:* symbol, or decimal digits 0-16. *reg3:* symbol, or decimal digits 0-16. **Default** for *reg1:* 0 **Default** for *reg2:* 1 **Default** for *reg3:* 2

The parameters are explained below:

byte addr
specifies the address of the byte to which the OR function is to be applied.

,mask
specifies the value to be ORed to the byte at the address specified above.

,REF=stor addr
specifies the address of a storage location on a fullword boundary. This address provides the means by which the COMPARE AND SWAP instruction may be executed. The address must be less than or equal to the byte address specified above, and the difference between the addresses must be less than 4096. The two addresses must be addressable via the same base register.

,WREGS = *(reg1,reg2,reg3)*
,WREGS = *reg1,reg2)*
,WREGS = *(reg1,,reg3)*
,WREGS = *(,reg2,reg3)*
,WREGS = *(reg1)*
,WREGS = *(,reg2)*
,WREGS = *(,,reg3)*
specifies the work registers to be used to perform the COMPARE AND SWAP instruction. *reg1* is used to contain the 'old' byte; *reg2* is used to contain the 'updated' byte; and *reg3* is used to contain the mask.

Example 1

Operation: Provide a lock on the byte of storage specified by the address SRTESTAT. UCBOB is the address used to reference the byte, and SRTECHGS is the mask used.

```
    OIL SRTESTAT,SRTECHGS,WREGS=(15,4,5),REF=UCBOB
```

PGFIX — Fix Virtual Storage Contents

The PGFIX macro instruction makes virtual storage areas resident in real storage and ineligible for page-out while the requesting task's address space occupies real storage. The PGFIX function is available only to authorized system functions and users.

PGFIX does not prevent pages from being paged out when an entire address space is swapped out of real storage. Consequently, when using the PGFIX macro instruction, you can not assume a constant real address mapping for fixed pages that are susceptible to swapping.

The standard form of the PGFIX macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede PGFIX.
PGFIX	
ƀ	One or more blanks must follow PGFIX.
R	
,A=*start addr*	*start addr:* A-type address, or register (1) or (2) - (12).
,ECB=*ecb addr*	*ecb addr:* A-type address, or register (0) or (2) - (12).
,EA=*end addr*	*end addr:* A-type address, or register (2) - (12) or (15). **Default:** *start addr* + 1
,LONG=Y ,LONG=N	**Default:** LONG=Y
,RELEASE=N ,RELEASE=Y	**Default:** RELEASE=N **Note:** RELEASE=Y may only be specified with EA above.
,RELATED=*value*	*value*: any valid macro keyword specification.

The parameters are explained below:

R
specifies that no parameter list is being supplied with this request.

,A=*start addr*
specifies the start address of the virtual area to be fixed.

,ECB=*ecb addr*
specifies the address of the ECB that is used to signal event completion.
Note: If the user intends to wait on the ECB as part of an ECB list, he must ensure that the list and associated ECBs are fixed in real storage before issuing the WAIT. The PGFIX service routine ensures that the specified ECB is fixed.

,EA=*end addr*
specifies the end address + 1 of the virtual area to be fixed.

,LONG=Y
,LONG=N
specifies that the relative real time duration anticipated for the fix is long (Y) or short (N).

,RELEASE=N
,RELEASE=Y
specifies that the contents of the virtual area is to remain intact (N) or be released (Y).

,RELATED = *value*

specifies information used to self-document macro instructions by 'relating' functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

The RELATED parameter is available on macro instructions that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE, and on macro instructions that relate to previous occurrences of the same macro instructions (for example, CHAP and ESTAE).

The parameter may be used, for example, as follows:

```
GET1     GETMAIN     R,LV=4096,RELATED=(FREE1,'GET STORAGE')
FREE1    FREEMAIN    R,LV=4096,A=(1),RELATED=(GET1,'FREE STORAGE')
```

When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	**Meaning**
00	Operation completed normally; ECB posted complete.
04	Operation abnormally terminated. Operation incomplete because of invalid address in virtual subarea list entry; ECB posted complete.
08	Operation proceeding; ECB will be posted when all requested pages are fixed in real storage.
10	Operation abnormally terminated. Virtual subarea list entry or ECB address invalid; no ECB is posted.

The ECB is unchanged if the request was initiated but not complete (return code 8), or if an ABEND was issued with return code 10. Otherwise, the ECB is posted complete with code:

0 - operation completed successfully.
4 - operation incomplete because of invalid address in VSL entry.

If the return code issued is 8, the ECB is posted asynchronously when paging I/O has completed, with code:

0 - operation completed successfully.
4 - operation incomplete because of paging error; requesting TCB will be abnormally terminated.

The ECB code is posted in the low-order 3 bytes of the ECB, and is right-justified.

Incompatible Parameters

The following parameters were valid in Release 1 of OS/VS2, but are not supported in MVS:

SUSPEND = N, will be ignored.
SUSPEND = Y, will be ignored.
ECBIND = address, will probably cause errors.

Example 1

Operation: Fix a single byte of virtual storage addressed by register 3. Note that the full 4096-byte page containing the specified byte will actually be fixed. The storage will be long fixed.

```
PGFIX    R,A=(R3),ECB=(R5)
```

Example 2

Operation: Fix virtual storage without using a virtual subarea list. Storage will be long fixed.

```
PGFIX    R,A=(R3),EA=(R4),ECB=ECB1
```

Example 3

Operation: Fix, but not long-fix, virtual storage, and ensure that the pages fully included in the address range are to be forfeited before fixing the area specified by registers 3 and 4.

```
PGFIX    R,A=(R3),EA=(R4),ECB=(R5),LONG=N,RELEASE=Y
```

PGFIX (List Form)

The list form of the PGFIX macro instruction uses a virtual subarea list.

The list form of the PGFIX macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b̸	One or more blanks must precede PGFIX.
PGFIX	
b̸	One or more blanks must follow PGFIX.
L	
,LA=*list addr*	*list addr:* A-type address, or register (1) or (2) - (12).
,ECB=*ecb addr*	*ecb addr:* A-type address, or register (0) or (2) - (12).
,LONG=N ,LONG=Y	**Default:** LONG=N
,RELEASE=N ,RELEASE=Y	**Default:** RELEASE=N
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained under the standard form of the PGFIX macro instruction, with the following exceptions:

L
specifies that a parameter list is being supplied with this request.

,LA=*list addr*
specifies the address of the first entry of a virtual subarea list.

Example 1

Operation: Fix virtual storage, providing a virtual subarea list addressed by register 5.

```
PGFIX    L,LA=(R5),ECB=(R6)
```

PGFREE — Free Virtual Storage Contents

The PGFREE macro instruction makes virtual storage areas that fixed via the PGFIX macro instruction eligible for page-out. The PGFREE function is available only to authorized system functions and users.

Note that a fixed page is not considered pageable until the number of PGFREEs issued for the page is equal to the number of PGFIXes previously issued for that page. That is, a page is not automatically removed from real storage as the result of the issuance of a PGFREE macro instruction.

The standard form of the PGFREE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede PGFREE.
PGFREE	
ƀ	One or more blanks must follow PGFREE.
R	
,A=*start addr*	*start addr:* A-type address, or register (1) or (2) - (12).
,ECB=*ecb addr*	*ecb addr:* A-type address, or register (0) or (2) - (12).
,EA=*end addr*	*end addr:* A-type address, or register (2) - (12) or (15). **Default:** *start addr* + 1
,RELEASE=N ,RELEASE=Y	**Default:** RELEASE=N **Note:** RELEASE=Y may only be specified with EA above.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained below:

R
specifies that no parameter list is being supplied with this request.

,A=*start addr*
specifies the start address of the virtual area to be freed.

,ECB=*ecb addr*
specifies the address of the ECB that was used in a prior PGFIX request. This parameter is used if there is any possibility that the ECB for the previously issued PGFIX was not posted complete.

,EA=*end addr*
specifies the end address + 1 of the virtual area to be freed.

,RELEASE=N
,RELEASE=Y
specifies that the contents of the virtual area is to remain intact (N) or be released (Y).

,RELATED=*value*
specifies information used to self-document macro instructions by 'relating' functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

The RELATED parameter is available on macro instructions that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE, and on macro instructions that relate to previous occurrences of the same macro instructions (for example, CHAP and ESTAE).

The parameter may be used, for example, as follows:

```
GET1     GETMAIN     R,LV=4096,RELATED=(FREE1,'GET STORAGE')
FREE1    FREEMAIN    R,LV=4096,A=(1),RELATED=(GET1,'FREE STORAGE')
```

When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Operation completed normally.
04	Operation abnormally terminated. Operation incomplete because of invalid address in virtual subarea list entry.
10	Operation abnormally terminated. Virtual subarea list entry or ECB address invalid.

Incompatible Parameters

The following parameters were valid in Release 1 of OS/VS2, but are not supported in MVS:

ECBIND = address, will probably cause errors.

Example 1

Operation: Free the storage in Example 1 of standard-form PGFIX.

```
PGFREE  R,A=(R3)
```

Example 2

Operation: Free the storage in Example 2 of standard-form PGFIX.

```
PGFREE  R,A=(R3),EA=(R4)
```

Example 3

Operation: Free the storage in Example 3 of standard-form PGFIX, and forfeit the pages full included in the address range.

```
PGFREE  R,A=(R3),EA=(R4),ECB=(R5),RELEASE=Y
```

PGFREE (List Form)

The list form of the PGFREE macro instruction uses a virtual subarea list.

The list of the PGFREE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede PGFREE.
PGFREE	
ƀ	One or more blanks must follow PGFREE.
L	
,LA=*list addr*	*list addr:* A-type address, or register (1) or (2) - (12).
,ECB=*ecb addr*	*ecb addr:* A-type address, or register (0) or (2) - (12).
,RELEASE=N ,RELEASE=Y	**Default:** RELEASE=N
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained under the standard form of the PGFREE macro instruction, with the following exceptions:

L
specifies that a parameter list is being supplied with this request.

,LA = *list addr*
specifies the address of the first entry of a virtual subarea list.

Example 1

Operation: Free the storage in Example 1 of list-form PGFIX.

```
PGFREE  L,LA=(R5)
```

POST — Signal Event Completion

The POST macro instruction is described in the **OS/VS2 Supervisor Services and Macro Instructions,** with the exception of the ASCB and ERRET parameters. These parameters are restricted in use and should only be used with tasks in supervisor state, APF-authorized, or with protection key 0-7.

The syntax of the complete POST macro instruction is shown below. However, only the explanation of the restricted parameters is presented. Explanation of the other parameters can be found in **OS/VS2 Supervisor Services and Macro Instructions.**

The standard form of the POST macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede POST.
POST	
ƀ	One or more blanks must follow POST.
ecb addr	*ecb addr:* RX-type address, or register (2) - (12).
,comp code	*comp code:* symbol, decimal or hexadecimal digit, or register (0) or (2) - (12). **Range of values:** 0 - 2^{30}-1 **Default:** 0
,ASCB=*addr*,ERRET=*err addr*	*addr:* RX-type address, or register (2) - (12). *err addr:* RX-type address, or register (2) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters restricted in use are explained below. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

,ASCB=*addr*,ERRET=*err addr*
specifies the address of the ASCB of the address space containing the ECB being posted, and the address of the routine to be given control when an error condition resulting from a POST failure is detected.

Example 1

Operation: Post an event control block whose address is ECB, where the address space containing the ECB has an ASCB specified by register 5, and where ERRRTN is the routine to be given control on error conditions.

```
POST     ECB,ASCB=(REG5),ERRET=ERRRTN
```

Example 2

Operation: Post the ECB from example 1 with a hexadecimal completion code of 3FF.

```
POST     ECB,X'3FF',ASCB=(REG5),ERRET=ERRRTN
```

POST (List Form)

The list form of the POST macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede POST.
POST	
ƀ	One or more blanks must follow POST.
ecb addr	*ecb addr:* A-type address.
,ASCB=*addr*,ERRET=*err addr*	*addr:* A-type address. *err addr:* A-type address.
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

The parameters are explained under the standard form of the POST macro instruction, with the following exceptions:

,MF=L
: specifies the list form of the POST macro instruction.

POST (Execute Form)

The execute form of the POST macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede POST.
POST	
ƀ	One or more blanks must follow POST.
ecb addr	*ecb addr:* RX-type address, or register (2) - (12).
,comp code	*comp code:* symbol, decimal or hexadecimal digit, or register (0) or (2) - (12). **Range of values:** 0 - 2^{30}-1
,ASCB=*addr*,ERRET=*err addr*	*addr:* RX-type address, or register (2) - (12). *err addr:* RX-type address, or register (2) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E *,prob addr*)	*prob addr:* RX-type address, or register (1) or (2) - (12).

The parameters are explained under the standard form of the POST macro instruction, with the following exceptions:

,MF=(E *,prob addr*)
specifies the execute form of the POST macro instruction using a remote control program parameter list.

PURGEDQ — Purge SRB Activity

The PURGEDQ macro instruction provides the facility for a task to purge particular SRB activity. Because an SRB routine is dispatched asynchronously to the actual issuance of a SCHEDULE macro instruction, the conditions that existed in the system at the time the SCHEDULE was issued may have totally changed by the time the routine is dispatched. If, in this time interval, the environment that the asynchronous routine requires to run successfully has been changed, the results would be unpredictable. For this reason, the PURGEDQ macro instruction is available to:

- Dequeue SRBs not yet dispatched.
- Dequeue or allow completed processing for SRBs previously scheduled.
- 'Clean up' each dequeued SRB.

The standard form of the PURGEDQ macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede PURGEDQ.
PURGEDQ	
ƀ	One or more blanks must follow PURGEDQ.
RMTR=*RMTR addr*	*RMTR addr:* RX-type address, or register (2) - (12).
,ASID=*ASID addr*	*ASID addr:* RX-type address, or register (2) - (12).
,ASIDTCB=*TCB addr*	*TCB addr:* RX-type address, or register (2) - (12).

The parameters are explained below:

RMTR=*RMTR addr*
specifies the address of the resource manager termination routine.

,ASID=*ASID addr*
specifies the address of a halfword containing the address space identifier. PURGEDQ will search for SRBs scheduled to be dispatched into the address space specified by ASID.

,ASIDTCB=*TCB addr*
specifies the address of a doubleword in the following format:

bytes 0-1	Reserved
bytes 2-3	ASID or zero
bytes 4-7	TCB address or zero

Example 1

Operation: All SRBS scheduled into the current address space and related to the current (terminating) task are to be purged by the RMTR routine IEAVRSPG.

```
PURGEDQ RMTR=IEAVRSPG
```

PURGEDQ (List Form)

The list form of the PURGEDQ macro instruction is used to construct a remote program parameter list.

The list form of the PURGEDQ macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede PURGEDQ.
PURGEDQ	
ƀ	One or more blanks must follow PURGEDQ.
RMTR=*RMTR addr*	*RMTR addr:* A-type address.
,ASID=*ASID addr*	*ASID addr:* A-type address.
,ASIDTCB=*TCB addr*	*TCB addr:* A-type address.
,MF=L	

The parameters are explained under the standard form of the PURGEDQ macro instruction, with the following exceptions:

,MF=L
specifies the list form of the PURGEDQ macro instruction.

PURGEDQ (Execute Form)

The execute form of the PURGEDQ macro instruction uses a remote control program parameter list. The parameter list is constructed using the list form of PURGEDQ.

The execute form of the PURGEDQ macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede PURGEDQ.
PURGEDQ	
ƀ	One or more blanks must follow PURGEDQ.
RMTR=*RMTR addr*	*RMTR addr:* RX-type address, or register (2) - (12).
,ASID=*ASID addr*	*ASID addr:* RX-type address, or register (2) - (12).
,ASIDTCB=*TCB addr*	*TCB addr:* RX-type address, or register (2) - (12).
,MF=(E, *ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

The parameters are explained under the standard form of the PURGEDQ macro instruction, with the following exceptions:

,MF=(E, *ctrl addr*)
specifies the execute form of the PURGEDQ macro instruction, using a remote control program parameter list.

Example 1

Operation: All SRBs scheduled into the address space designated by register 6 are to be purged by the RMTR routine IEAVRSPG. Register 1 is a pointer to the parameter list, and register 7 indicates that all SRBs are to be purged.

```
PURGEDQ MF=(E,R1)),ASID=(R6),ASIDTCB=(R7),RMTR=IEAVRSPG
```

QEDIT — Command Input Buffer Manipulation

The QEDIT macro instruction generates the required entry parameters and processes the command input buffer for the following uses:

- Dechaining and freeing of a command input buffer (CIB) from the CIB chain for a task.
- Setting a limit for the number of CIBs that may be simultaneously chained for a task.

The QEDIT macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b̸	One or more blanks must precede QEDIT.
QEDIT	
b̸	One or more blanks must follow QEDIT.
ORIGIN=*CIB addr ptr*	*CIB addr ptr:* RX-type address, or register (2) - (12).
,BLOCK=*CIB addr*	*CIB addr:* RX-type address, or register (2) - (12).
,CIBCTR=*CIB nmbr*	*CIB nmbr:* decimal digit, with a maximum value of 255.

The parameters are explained below:

ORIGIN=*CIB addr ptr*
specifies the address of the pointer to the first CIB chain for the task. This address is obtained using the EXTRACT macro instruction. If ORIGIN is the only parameter specified, the caller must be executing under system key 0-7; in this case, the entire CIB chain will be freed.

,BLOCK=*CIB addr*
specifies the address of the CIB that is to be freed from the CIB chain for a task.

,CIBCTR=*CIB nmbr*
specifies the limit for the number of CIBs to be chained at any time for a task.

Example 1

Operation: Free the entire CIB chain, where register 8 contains the address of the pointer to the CIB chain.

```
QEDIT   ORGIN=(8)
```

Example 2

Operation: Free the CIB whose address is in register 5 from the CIB chain. Register 8 contains the address of the pointer to the CIB chain.

```
QEDIT   ORIGIN=(8),BLOCK=(5)
```

RESERVE — Reserve a Device (Shared DASD)

The RESERVE macro instruction is used to reserve a device for use by a particular system; it must be issued by each task needing device reservation. The RESERVE macro instruction protects the issuing task from interference by other tasks in the system and locks out the other CPU. When the reserving program no longer needs the reserved device, it should issue a DEQ macro instruction, specifying the UCB parameter, to release the device. If a task issues two RESERVE instructions for the same resource without an intervening DEQ an abnormal termination will result unless the second RESERVE specifies the keyword parameter RET= or ECB=. (If a restart occurs when a RESERVE is in effect for devices, the system will not restore the RESERVE; the user's program must reissue the RESERVE.) If a DEQ is not issued for a particular device, termination routines will release devices reserved by a terminating task.

To use the shared DASD option in higher level languages, an assembler language subroutine should be written to issue the RESERVE macro instruction. The following information should be passed to this routine: ddname, qnameaddress, rnameaddress, rnamelength, and RET parameter.

The standard form of the RESERVE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede RESERVE.
RESERVE	
ƀ	One or more blanks must follow RESERVE.
(	
qname addr	*qname addr:* A-type address, or register (2) - (12).
,*rname addr*	*rname addr:* A-type address, or register (2) - (12).
, ,E ,S	**Default:** E
, ,*rname length*	*rname length:* symbol, decimal digit, or register (2) - (12).
,SYSTEMS	
)	
,RET=TEST ,RET=USE ,RET=HAVE	
,ECB=*ecb addr*	*ecb addr:* A-type address, or register (2) - (12).
,UCB=*ucb addr*	*ucb addr:* A-type address, or register (2) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained below:

(
specifies the beginning of the resource description.

qname addr
specifies the address in virtual storage of an 8-character name. The name should not start with SYS, so that it will not conflict with system names. Every task issuing RESERVE against the same resource must use the same *qname* and *rname* to represent the resource.

,rname addr

specifies the address in virtual storage of the name used in conjunction with *qname* to represent a single resource. The name can be qualified, and must be from 1 to 255 bytes long.

,

,E

,S

specifies whether the request is for exclusive (E) or shared (S) control of the resource. If the resource is modified while under control of the task, the request must be for exclusive control; if the resource is not modified, the request should be for shared control.

,

,rname length

specifies the length of the *rname* described above. If this parameter is omitted, the assembled length of the *rname* is used. You can specify a value between 1 to 255 to override the assembled length, or you may specify a value of 0. If 0 is specified, the length of the *rname* must be contained in the first byte at the *rname addr* specified above.

,SYSTEMS

specifies that the resource is shared between systems.

)

specifies the end of the resource description.

,RET=TEST

,RET=USE

,RET=HAVE

specifies a conditional request for all the resources name above.

RET=TEST the availability of the resources is to be tested, but control of the resources is not requested.

RET=USE control of the resources is to be assigned to the active task only if the resources are immediately available.

RET=HAVE control of the resources is requested only if a request has not been made previously for the same task.

,ECB=*ecb addr*

specifies the address of an ECB, and conditionally requests the resource named in the macro instruction.

,UCB=*ucb addr*

specifies the address of a fullword that contains the address of the UCB for the device to be reserved.

,RELATED=*value*

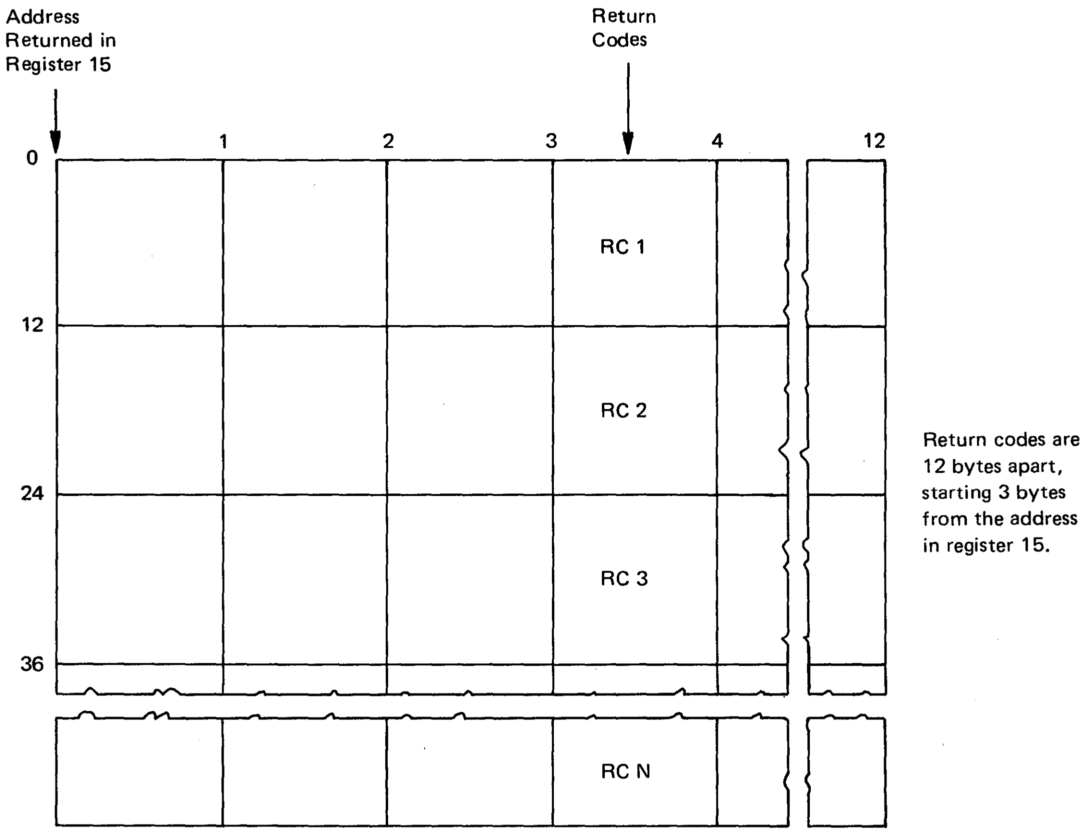
specifies information used to self-document macro instructions by 'relating' functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

The RELATED parameter is available on macro instructions that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE, and on macro instructions that relate to previous occurrences of the same macro instructions (for example, CHAP and ESTAE).

The parameter may be used, for example, as follows:

```
GET1     GETMAIN     R,LV=4096,RELATED=(FREE1,'GET STORAGE')
FREE1    FREEMAIN    R,LV=4096,A=(1),RELATED=(GET1,'FREE STORAGE')
```

Return codes are provided by the control program only if you specify RET=TEST, RET=USE, RET=HAVE, or ECB=; otherwise, return of the task to the active condition indicates that control of the resource has been assigned to the task. If return code for the resource named in the RESERVE macro instruction is 0, register 15 contains 0. If the return code is not 0, register 15 contains the address of a storage area containing the return codes, as shown in Figure 29.



Address Returned in Register 15
Return Codes
0
1
2
3
4
12
12
24
36
RC 1
RC 2
RC 3
RC N
Return codes are 12 bytes apart, starting 3 bytes from the address in register 15.

Figure 29. Return Code Area Used by RESERVE

The return code is placed in the parameter list resulting from the macro expansion. The return codes are shown below.

Hexadecimal Code	Meaning
0	For RET=TEST, the resource was immediately available. For RET=USE, RET=HAVE, or ECB=, control of the resource has been assigned to the active task.
4	For RET=TEST or RET=USE, the resource is not immediately available. For ECB=, the ECB will be posted when available.
8	A previous request for control of the same resource has been made for the same task. Task has control of resource. If bit 3 is on — shared control of resource; if bit 3 is off — exclusive control.
20	A previous request for control of the same resource has been made for the same task. Task does not have control of resource.

Example 1

Operation: Unconditionally reserve exclusive control of a device. The length of the rname is allowed to default.

```
RESERVE (MAJOR3,MINOR3,E,,SYSTEMS),UCB=(R3)
```

RESERVE (List Form)

The list form of the RESERVE macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede RESERVE.
RESERVE	
ƀ	One or more blanks must follow RESERVE.
(	
qname addr	*qname addr:* A-type address.
, ,*rname addr*	*rname addr:* A-type address.
, ,E ,S	
, ,*rname length*	*rname length:* symbol or decimal digit. **Note:** *rname length* must be coded if a register is specified for *rname addr* above.
, ,SYSTEMS	
)	
,RET=TEST ,RET=USE ,RET=HAVE	
,ECB=*ecb addr*	*ecb addr:* A-type address.
,UCB=0	
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

The parameters are explained under the standard form of the RESERVE macro instruction, with the following exceptions:

,MF = L
: specifies the list form of the RESERVE macro instruction.

RESERVE (Execute Form)

The execute form of the RESERVE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede RESERVE.
RESERVE	
ƀ	One or more blanks must follow RESERVE.
(	**Note:** (and) are the beginning and end of a parameter list. The entire list is optional. If nothing in the list is desired, the (,), and all parameters between (and) should not be specified. If something in the list is desired, then (,), and all parameters in the list should be specified as indicated at the left.
qname addr	*qname addr:* RX-type address, or register (2) - (12).
, ,*rname addr*	*rname addr:* RX-type address, or register (2) - (12).
, ,E ,S	
, ,*rname length*	*rname length:* symbol, decimal digit, or register (2) - (12).
, ,SYSTEMS	
)	
,RET=TEST ,RET=USE ,RET=HAVE	
,ECB=*ecb addr*	*ecb addr:* RX-type address, or register (2) - (12).
,UCB=*ucb addr*	*ucb addr:* RX-type address, or register (2) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E, *ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

The parameters are explained under the standard form of the RESERVE macro instruction, with the following exceptions:

,MF=(E, *ctrl addr*)
specifies the execute form of the RESERVE macro instruction using a remote control program parameter list.

RISGNL — Issue Remote Immediate Signal

The RISGNL macro instruction uses the emergency signal (ES) function of the signal processor (SIGP) instruction to invoke the execution of a specified software program on one of the CPUs in a tightly coupled multiprocessing system. The program may be requested to execute in parallel or serially with the function requesting the program.

Ten of the twelve SIGP hardware functions are defined as direct services and are accessible via the DSGNL macro instruction. The other SIGP function is accessible via the RPSGNL macro instruction.

The RISGNL macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede RISGNL.
RISGNL	
ƀ	One or more blanks must follow RISGNL.
PARALLEL SERIAL	
,CPU=*PCCA addr*	*PCCA addr:* RX-type address, or register (1).
,EP=*entry name addr*	*entry name addr:* RX-type address, or register (12).
,PARM=*parm addr*	*parm addr:* RX-type address, or register (11).

The parameters are explained below:

PARALLEL
SERIAL
specifies that control is to be returned to the caller when the specified receiving routine has been given control (PARALLEL) or has completed execution (SERIAL) on the designated CPU.

,CPU=*PCCA addr*
specifies the address of the physical configuration communication area (PCCA) of the CPU on which the function is to be performed.

,EP=*entry name addr*
specifies the address of the entry name of the receiving routine to be executed on the specified CPU.

,PARM=*parm addr*
specifies the address of a user-defined fullword parameter to be passed to the receiving routine.

When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	**Meaning**
00	Specified receiving routine has been given control or has completed execution, as requested.
04	Function not initiated because addressed CPU not online.
08	Function unsuccessful. Emergency signal could not be generated on CPU. Status returned in register 0.
12	Function unsuccessful. Specified CPU is either not installed, not configured into system, or powered off.
16	CPU is a uniprocessor and does not have signal processor sending and receiving capabilities.
20	CPU alive bit was turned off during the remote immediated window spin routine.

With a return code of 8, register 0 contains:

Bits	**Meaning**
0	Equipment check
1-24	Reserved
25	Stopped
26	Operator intervening
27	Check stop
28	Not ready
29	Reserved
30	Invalid function
31	Receiver check

Example 1

Operation: The routine whose address is in register 12 is to be given control on the CPU whose PCCA address is in register 1. The routine will execute in parallel with the caller who invoked RISGNL.

```
RISGNL  PARALLEL,CPU=(1),EP=(12)
```

Example 2

Operation: The routine whose address is in register 12 is to be given control on the CPU whose PCCA address is in register 1. The routine will complete before the caller of RISGNL receives control again. Register 11 will contain the address of a parameter to be passed.

```
RISGNL  SERIAL,CPU=(1),EP=(12),PARM=(11)
```

RPSGNL — Issue Remote Pendable Signal

The RPSGNL macro instruction uses the external call (EC) function of the signal processor (SIGP) instruction to invoke the execution of one of six software programs on one of the CPUs in a tightly coupled multiprocessing system.

Ten of the twelve SIGP hardware functions are defined as direct services and are accessible via the DSGNL macro instruction. The other SIGP function is accessible via the RISGNL macro instruction.

The RPSGNL macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede RPSGNL.
RPSGNL	
ƀ	One or more blanks must follow RPSGNL.
SWITCH SIO RQCHECK GTFCRM MODE MF1TCH	
,CPU=*PCCA addr*	*PCCA addr:* RX-type address, or register (1).

The parameters are explained below:

SWITCH
SIO
RQCHECK
GTFCRM
MODE
MF1TCH
specifies the action to be performed:

SWITCH Memory/task switch function

SIO IOS start I/O function

RQCHECK Timer supervision TQE check function, to ensure that TQE in real time queue is being timed.

GTFCRM GTF function, to modify monitor call control registers

MODE RMS function, to modify RMS-oriented control registers

MF1TCH MF1 function, to issue TCH instructions on CPU to which channels are attached.

,CPU = *PCCA addr*
specifies the address of the physical configuration communication area (PCCA) of the CPU on which the function is to be executed.

When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Specified CPU is online and has been notified that the specified service is to be executed.
04	Function not initiated because addressed CPU not online.
08	Function unsuccessful. External call signal could not be generated on CPU. Status returned in register 0.
12	Specified CPU is either not installed, not configured into system, or powered off.
16	CPU is a uniprocessor and does not have signal processor sending and receiving capabilities.

With a return code of 8, register 0 contains:

Bits	Meaning
0	Equipment check
1-25	Reserved
26	Operator intervening
27	Check stop
28	Not ready
29-30	Reserved
31	Receiver check

Example 1

Operation: The service routine of memory switch is to be given control on the CPU whose PCCA address is in register 1.

```
RPSGNL   SWITCH,CPU=(1)
```

Example 2

Operation: The IOS start I/O routine is to be given control on the CPU whose PCCA address is in register 1.

```
RPSGNL   SIO,CPU=(1)
```

SCHEDULE — Schedule System Services for Asynchronous Execution

The SCHEDULE macro instruction schedules system services for asynchronous execution. These services may be scheduled for execution in any address space and may be scheduled at either global or local priorities.

Services scheduled at a global priority will have a priority that is greater than, and independent of, any address space priority. Services scheduled at a local priority will have the priority of the specific address space they execute in, but will still have a priority greater than that of any task within the address space. To use SCHEDULE you must be in supervisor state, key zero.

Note: This macro instruction requires that the SRB and CVT mapping macros be assembled into the caller's csect.

The SCHEDULE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede SCHEDULE.
SCHEDULE	
ƀ	One or more blanks must follow SCHEDULE.
SRB=*SRB addr*	*SRB addr:* RX-type address, or register (1) or (2) - (12).
,SCOPE=LOCAL ,SCOPE=GLOBAL	**Default:** SCOPE=LOCAL.

The parameters are explained below:

SRB=*SRB addr*
specifies the address of the service request block (SRB).

,SCOPE=LOCAL
,SCOPE=GLOBAL
specifies whether the service is to be scheduled at a local or global priority.

Example 1

Operation: Schedule an SRB at a global priority.

```
SCHEDULE     SRB=(1),SCOPE=GLOBAL
```

Example 2

Operation: Schedule an SRB at a local priority.

```
SCHEDULE     SRB=(1),SCOPE=LOCAL
```

SDUMP — Dump Virtual Storage

The SDUMP macro instruction provides a dumping capability for the system routines. It invokes SVC DUMP to provide a fast unformatted dump of virtual storage to a data set. It is intended to be used by system routines that suffer errors.

SVC DUMP is available only to authorized programs. Issuers of SDUMP with entry by SVC must be authorized via APF or have a control program key. Branch entry callers must be key 0, supervisor state, and must be in SRB mode, or own a lock, or be disabled (with supervisor bit on).

The service of initiating an SVC DUMP in any address space is provided for callers who need to dump an address space other than the one in which they are running. A branch entry to this service is also provided for callers who wish a dump of their own or another address space but cannot issue an SVC.

The standard form of the SDUMP macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must follow SDUMP.
SDUMP	
ƀ	One or more blanks must follow SDUMP.

HDR=*'dump title'*
HDRAD=*dump title addr*
	dump title: from 1 to 100 characters.
	dump title addr: A-type address, or register (2) - (12).

,DCB=*dcb addr*	*dcb addr:* A-type address, or register (2) - (12).
,ASID=*ASID addr*	*ASID addr:* A-type address, or register (2) - (12).
,ECB=*ecb addr*	*ecb addr:* A-type address, or register (2) - (12).
,SDATA=*(data code)*	*data code:* any combination of the following, separated by commas: SQA RGN ALLPSA LPA PSA TRT NUC CSA LSQA SWA
,STORAGE=*(strt addr,end addr)* ,LIST=*list addr*	*strt addr:* A-type address, or register (2) - (12). *end addr:* A-type address, or register (2) - (12). *list addr:* A-type address, or register (2) - (12). **Note:** One or more pairs of addresses may be specified, separated by commas. For example: ,STORAGE=*(strt addr,end addr,strt addr,end addr)*
,BUFFER=NO ,BUFFER=YES	**Default:** BUFFER=NO
,QUIESCE=YES ,QUIESCE=NO	**Default:** QUIESCE=YES
,BRANCH=NO ,BRANCH=YES	**Default:** BRANCH=NO **Note:** If BRANCH is specified, ASID must also be specified.

The parameters are explained below:

HDR=*'dump title'*
HDRAD=*dump title addr*
specifies the title or address of the title to be used for the dump. If HDR is specified, the title must appear enclosed in apostrophes, although the apostrophes do not appear in the actual title. If HDRAD is specified, the first byte at the indicated address specifies the length of the title in bytes.

,DCB=*dcb addr*
specifies the address of a previously opened data control block for the data set that is to contain the dump. If this parameter is omitted, one of the SYS1.DUMP data sets will be used.

,ASID=*ASID addr*
specifies the address of a halfword containing the address space identifier of the address space to be dumped. If this parameter is omitted, the current address space will be dumped. If 0 is specified, the dump is scheduled in the current address space.

,ECB=*ecb addr*
specifies the address of a fullword containing the address of an event control block that is posted on completion of a scheduled dump. If this parameter is omitted, the caller is not notified of the completion of the scheduled dump.

,SDATA=*(data code)*
specifies the system control program information to be dumped:
SQA — The system queue area.
ALLPSA — All of the prefixed storage areas in the system.
PSA — The prefixed storage area for the current CPU.
NUC — The nucleus.
LSQA — The local system queue area for the address space being dumped.
RGN — The allocated pages in the private area of the address space being dumped. This includes the LSQA and the SWA.
LPA — The active link pack area modules and SVCs for the address space being dumped.
TRT — The GTF trace buffers if GTF tracing is active, or the supervisor trace table if it is not active. If a dump occurs in a GTF address space, no attempt will be made to include trace information.
CSA — The common service area subpools.
SWA — The scheduler work area subpools in the address space being dumped.

,STORAGE=*(strt addr,end addr)*
,LIST=*list addr*
specifies one or more pairs of starting and ending address or a list of starting and ending addresses of areas to be dumped. (Each starting address must be less than its corresponding ending address.) The storage list must contain an even number of addresses, and each address must occupy one fullword. In the list, the high order bit of the fullword containing the last ending address of the list must be set to 1; all other high order bits must be set to 0.

,BUFFER=NO
,BUFFER=YES
specifies that the contents of the SQA buffer is (YES) or is not (NO) to be included in the dump. (The SQA buffer does not include the SDUMP parameter list or any data pointed to by the parameter list.)

,QUIESCE = YES
,QUIESCE = NO

specifies that the system is to be set nondispatchable until the contents of the SQA and the CSA are dumped (YES), or that the system is to be left dispatchable (NO). If SDATA parameter does not specify SQA or CSA, the QUIESCE=YES request is ignored.

,BRANCH = NO
,BRANCH = YES

specifies that a branch entry is to be used for interfacing with SVC DUMP to schedule a dump (YES), or that an SVC 51 instruction is to be generated for interfacing with SVC DUMP. This parameter can only be used by key 0, supervisor state routines that are in SRB mode, locked, or disabled to schedule a dump.

If the ASID parameter was not specified, register 15 contains one of the following return codes when control is returned:

Hexadecimal Code	Meaning
00	A complete dump was taken.
04	A partial dump was taken.
08	The system was unable to take a dump.

If the ASID parameter was specified, register 15 contains one of the following return codes when control is returned:

Hexadecimal Code	Meaning
00	A dump was scheduled. If an ECB was supplied, it will be posted on completion of the dump.
08	The system was unable to schedule a dump.

If an ECB was supplied, one of the following codes is returned:

Hexadecimal Code	Meaning
00	A complete dump was taken.
04	A partial dump was taken.
08	The system was unable to take a dump.

Example 1

Operation: This example shows how SVC DUMP may be branch entered to initiate a dump in an address space for callers who cannot issue an SVC. Areas to be dumped are requested via three parameters (BUFFER, SDATA, and STORAGE). The dump will have the title indicated in the HDR parameter, and the caller requests to be notified of the completion of the scheduled dump via the ECB parameter.

```
SDUMP    HDR='USER DATA FOR TEST A',DCB=TESTADCB,BUFFER=YES,
         ASID=TSTAASID,ECB=(8),QUIESCE=YES,BRANCH=YES,
         STORAGE=(A,B,C,D,(9),E),SDATA=(ALLPSA,PSA,NUC,SQA,LSQA,
         RGN,LPA,SWA,CSA)
```

SDUMP (List Form)

Use the list form of the SDUMP macro instruction to construct a control program parameter list. You can specify any number of storage addresses using the STORAGE parameter. Therefore, the number of starting and ending address pairs in the list form of SDUMP must be equal to the maximum number of addresses specified in the execute form of the macro instruction.

The list form of the SDUMP macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede SDUMP.
SDUMP	
ƀ	One or more blanks must follow SDUMP.
HDR=*'dump title'* ,HDRAD=*dump title addr*	*dump title:* from 1 to 100 characters. *dump title addr:* A-type address.
,DCB=*dcb addr*	*dcb addr:* A-type address.
,SDATA=*(data code)*	*data code:* any combination of the following, separated by commas: SQA RGN ALLPSA LPA PSA TRT NUC CSA LSQA SWA
,STORAGE=*(strt addr,end addr)* ,LIST=*list addr*	*strt addr:* A-type address. *end addr:* A-type address. **Note:** One or more pairs of addresses may be specified, separated by commas. For example: ,STORAGE=*(strt addr,end addr,strt addr,end addr)*
,BUFFER=NO ,BUFFER=YES	**Default:** BUFFER=NO
,QUIESCE=YES ,QUIESCE=NO	**Default:** QUIESCE=YES
,MF=L	

The parameters are explained under the standard form of the SDUMP macro instruction, with the following exceptions:

,MF=L
specifies the list form of the SDUMP macro instruction.

SDUMP (Execute Form)

A remote control program parameter list is referred to and can be modified by the execute form of the SDUMP macro instruction.

If you code one or more of the SDATA parameters on the execute form of the macro instruction, any SDATA parameters coded on the list form will be lost.

The execute form of the SDUMP macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede SDUMP.
SDUMP	One or more blanks must follow SDUMP.
ƀ	

HDR=*'dump title'*
HDRAD=*dump title addr*
	dump title: from 1 to 100 characters.
	dump title addr: RX-type address, or register (2) - (12).

,DCB=*dcb addr*	*dcb addr:* RX-type address, or register (2) - (12).
,ASID=*ASID addr*	*ASID addr:* RX-type address, or register (2) - (12).
,ECB=*ecb addr*	*ecb addr:* RX-type address, or register (2) - (12).
,SDATA=*(data code)*	*data code:* any combination of the following, separated by commas: SQA RGN ALLPSA LPA PSA TRT NUC CSA LSQA SWA
,STORAGE=*(strt addr,end addr)* ,LIST=*list addr*	*strt addr:* RX-type address, or register (2) - (12). *end addr:* RX-type address, or register (2) - (12). *list addr:* RX-type address, or register (2) - (12). **Note:** One or more pairs of addresses may be specified, separated by commas. For example: ,STORAGE=*(strt addr,end addr,strt addr,end addr)*
,BUFFER=NO ,BUFFER=YES	
,QUIESCE=YES ,QUIESCE=NO	
,BRANCH=NO ,BRANCH=YES	**Note:** If BRANCH is specified, ASID must also be specified.
,MF=(E, *ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

The parameters are explained under the standard form of the SDUMP macro instruction, with the following exceptions:

,MF=(E, *ctrl addr*)
specifies the execute form of the SDUMP macro instruction using a remote control program parameter list.

Example 1

Operation: The execute form is used to add (SDATA areas) and change (BUFFER and QUIESCE) options in the SDUMP parameter list. The list form of SDUMP was previously used to create the basic SDUMP parameter list located by register 1.

```
SDUMP    SDATA=(SQA,LPA),BUFFER=NO,QUIESCE=NO,MF=(E,(1))
```

SETFRR — Set Up Functional Recovery Routines

The SETFRR macro instruction provides control program functions with the ability to define their recovery in the FRR (functional recovery routine) LIFO stack which is used during processing of the system recovery manager. Each branch-entered control program function can use SETFRR to define its own unique recovery environment.

The SETFRR macro instruction can be used to add, delete, or replace FRRs in the LIFO stack, or to purge all FRRs in the stack. The macro instruction also optionally returns to the user the address of a parameter area that is eventually passed to the FRR when an error occurs. The parameter area can be initialized with information such as tracking data that may be useful to the FRR.

SETFRR has no external linkages and expands directly inline. Users of SETFRR must be key 0 supervisor state and must hold a lock or be in SRB mode. All SETFRR users must include the DSECTs for the FRR stack (via the IHAFRRS mapping macro instruction) and the PSA via the IHAPSA mapping macro instruction) prior to using the SETFRR macro instruction. In addition, all disabled, locked, and SRB routines which defined recovery must be key 0 supervisor state when using the SETFRR macro instruction. Note that it is necessary to copy IHAPSA from MODGEN into MACLIB.

The SETFRR macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede SETFRR.
SETFRR	
ƀ	One or more blanks must follow SETFRR.
A,FRRAD=*FRR addr* R,FRRAD=*FRR addr* D P	*FRR addr:* A-type address, or register (0) - (15).
,WRKREGS=*(reg1,reg2)*	*reg1:* decimal digits 1-15. *reg2:* decimal digits 1-15.
,PARMAD=*parm area addr*	*parm area addr:* A-type address, or register (1) - (15). **Note:** This parameter may only be specified with A or R above.
,CLEAR=*YES* ,CLEAR=*NO*	
,RELATED=*value*	*value:* any valid macro keyword specification.

The explanation of the parameter is as follows:

A,FRRAD=*FRRAD addr*
R,FRRAD=*FRRAD addr*
D
P

specifies the operation to be performed on the FRR LIFO stack:

A — an FRR address is to be added to the stack.
R — the FRR address last added to the stack is to be replaced by another FRR address.
D — the FRR address last added to the stack is to be deleted.
P — all entries in the stack are to be purged.

FRRAD specifies the address of a fullword containing the FRR address that is to be added or replaced. The parameter specifies the FRR address in a register or specifies the address of a storage location containing the FRR address. If a register is specified for FRRAD it must be different than those specified for the WRKREGS keyword.

,WRKREGS=*(reg1,reg2)*

specifies two unique general purpose registers to be used as work registers in the code generated by the SETFRR macro expansion.

,PARMAD=*parm area addr*

specifies the address of a fullword to receive the address of the 24-byte parameter area provided by the system to the issuer of SETFRR. This parameter area is associated with the FRR address which has either been added to or has replaced an FRR address on the stack. This parameter area is passed to the FRR when an error occurs.

,CLEAR=*YES*
,CLEAR=*NO*

specifies whether or not the 24-byte parameter area provided by the macro instruction is to be cleared to zeros. When CLEAR=*NO* is specified, the macro expansion is reduced by 6 bytes and the execution time is reduced by 2/3. CLEAR= is valid only when PARMAD= is specified. The default value for the add (A) option is CLEAR=*YES*. The default value for the replace (R) option is CLEAR=*NO*.

,RELATED=*value*

specifies information used to self-document macro instructions by —relating— functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

The RELATED parameter is available on macro instructions that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE, and on macro instructions that relate to previous occurrences of the same macro instructions (for example, CHAP and ESTAE).

The parameter may be used, for example, as follows:

```
GET1     GETMAIN     R,LV=4096,RELATED=(FREE1,'GET STORAGE')
FREE1    FREEMAIN    R,LV=4096,A=(1),RELATED=(GET1,'FREE STORAGE')
```

Example 1

Operation: Add an FRR to the FRR stack, and return the address of the parameter list to the issuer of the SETFRR. The FRR address contained in register (R1) is placed on the FRR stack in the next available FRR entry. Register (R2) will contain the address of the parameter list associated with this FRR entry. Registers R3 and R4 are work registers used in the code generated by SETFRR in performing its operations.

```
SETFRR   A,FRRAD=(R1),PARMAD=(R2),WRKREGS=(R3,R4)
```

Example 2

Operation: Delete the last FRR added to the FRR stack.

```
SETFRR   D,WRKREGS=(1,6)
```

SETLOCK — Control Access to Serially Reusable Resources

The SETLOCK macro instruction is used to control access to serially reusable resources. Each kind of serially reusable resource is assigned a separate lock. To use SETLOCK, you must be executing in supervisor state with protection key 0. Also, SETLOCK users must include the DSECT for the PSA (via IHAPSA mapping macro) prior to using the SETLOCK macro instruction. Note that it is necessary to copy IHAPSA from MODGEN into MACLIB.

SETLOCK can be used to:

—Obtain a specified lock or set of locks.
—Release a specified lock or set of locks.
—Test a specified lock or set of locks to determine if the lock is held by the requesting CPU.

Two classes of locks exist: global and local. Two types of locks exist: spin and suspend. The descriptions of these locks and the hierarchy structure in which these locks are arranged are described under locking in this publication.

The OBTAIN option of SETLOCK macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede SETLOCK.
SETLOCK	
ƀ	One or more blanks must follow SETLOCK.
OBTAIN	
,TYPE=IOSCAT,ADDR=(11) ,TYPE=IOSUCB,ADDR=(11) ,TYPE=IOSLCH,ADDR=(11) ,TYPE=IOSYNCH,ADDR=(11) ,TYPE=ASM,ADDR=(11) ,TYPE=DISP ,TYPE=SALLOC ,TYPE=SRM ,TYPE=CMS ,TYPE=LOCAL	
,MODE=COND ,MODE=UNCOND ,MODE=UNCOND,DISABLED	**Note:** DISABLED may not be specified if TYPE=CMS or TYPE=LOCAL is specified above.
,REGS=SAVE ,REGS=USE	
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained below:

OBTAIN
specifies that the lockword is to be obtained or locked on the caller's behalf.

,TYPE = IOSCAT,ADDR = (11)
,TYPE = IOSUCB,ADDR = (11)
,TYPE = IOSLCH,ADDR = (11)
,TYPE = IOSYNCH,ADDR = (11)
,TYPE = ASM,ADDR = (11)
,TYPE = DISP
,TYPE = SALLOC
,TYPE = SRM
,TYPE = CMS
,TYPE = LOCAL

specifies the type of lock that is to be obtained on the caller's behalf.

ADDR = (11) specifies that the address of the lockword indicated by the TYPE parameter has been loaded into register 11 prior to the SETLOCK request.

IOSCAT IOS channel availability table lock. It is a global spin lock used by IOS to serialize access and updates to the channel availability table.

IOSUCB IOS unit control block lock. These locks (one per UCB) are global spin locks used to serialize access and updates to UCBs.

IOSLCH IOS logical channel queue lock. These locks (one per channel queue) are glocal spin locks used to serialize access and updates to the IOS logical channel queues.

IOSYNCH IOS synchronization lock. It is a global spin lock used to serialize the global IOS functions.

ASM Auxiliary storage manager lock. It is a global spin lock used to serialize use of the global ASM control blocks.

DISP Global dispatcher lock. It is a global spin lock used to serialize all functions associated with the dispatching of storage.

SALLOC Real storage manager and virtual storage manager space allocation lock. It is a global spin lock used to serialize the global functions of RSM and VSM.

SRM Systems resource manager lock. It is a global spin lock used to serialize use of the SRM control algorithms and associated data.

CMS Cross memory services lock. It is a global suspend lock used to serialize on more than one virtual storage where this serialization is not provided by one or more of the global locks.

LOCAL Storage in which lock of the storage the SETLOCK caller is executing. It is a local suspend lock used by supervisor functions which require serialization within that particular storage only.

,MODE = COND
,MODE = UNCOND
,MODE = UNCOND,DISABLED

specifies whether the lock is to be conditionally or unconditionally obtained.

COND specifies that the lock is to be conditionally obtained. That is, if the lock is not owned by another CPU, it will be acquired on the caller's behalf. If the lock is already held, control will be returned to the caller indicating that the lock is held and that either the caller already owns the lock or that another CPU or storage owns the lock.

UNCOND specifies that the lock is to be unconditionally obtained. That is, if the lock is not owned by another CPU, it will be acquired on the caller's behalf. If the lock is already held by the caller, control will be returned to the caller indicating that he already owns the lock. If the lock is held by another CPU, the caller's CPU will either spin on the lock until it is released or suspend the SETLOCK caller until the lock is released.

DISABLED specifies that the caller is already in a physically disabled state.

,REGS = SAVE
,REGS = USE
specifies the use of registers 11 through 1.

SAVE specifies that register contents are to be saved. Registers 11 through 14 will be saved in the area pointed to by register 13, and will be restored upon completion of the SETLOCK request. The savearea will consist of at least 5 words. Register 15 will contain the return code.

USE specifies that registers 14, 15, 0, and 1 are available for use. Registers 11, 12, and 13 will be saved in registers 15, 0, and 1, respectively, and will be restored upon completion of the SETLOCK request. Register 14 will be used as a link register; register 15 will contain the return code.

Note: If neither SAVE nor USE is specified, registers 11-14 are destroyed and register 13 contains the return code.

,RELATED = *value*
specifies information used to self-document macro instructions by 'relating' functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

The RELATED parameter is available on macro instructions that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE, and on macro instructions that relate to previous occurrences of the same macro instructions (for example, CHAP and ESTAE).

The parameter may be used, for example, as follows:

```
GET1    GETMAIN     R,LV=4096,RELATED=(FREE1,'GET STORAGE')
FREE1   FREEMAIN    R,LV=4096,A=(1),RELATED=(GET1,'FREE STORAGE')
```

When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	The lock was successfully obtained. The lock was free and is now held on the caller's behalf.
04	The lock was already held by the caller. The lockword id matches the caller's id.
08	The obtain process was unsuccessful. The lockword id does not match the caller's id.

Example 1

Operation: The global dispatcher lock DISP is to be conditionally requested. The RELATED parameter indicates that the DISP lock serializes the TCB resource, and the lock will either be freed at the location represented by NAME or SYM1 in module IEFVHA or by SYM2 in module IEFVFA.

```
SETLOCK OBTAIN,TYPE=DISP,MODE=COND,RELATED=(TCB,IEFVHA(NAME,
        SYM1),IEFVFA(SYM2))
```

The RELEASE option of the SETLOCK macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede SETLOCK.
SETLOCK	
ƀ	One or more blanks must follow SETLOCK.
RELEASE	
,TYPE=IOSCAT,ADDR=(11) ,TYPE=IOSUCB,ADDR=(11) ,TYPE=IOSLCH,ADDR=(11) ,TYPE=IOSYNCH,ADDR=(11) ,TYPE=ASM,ADDR=(11) ,TYPE=DISP ,TYPE=SALLOC ,TYPE=SRM ,TYPE=CMS ,TYPE=LOCAL ,TYPE=SPIN ,TYPE=ALL ,TYPE=*(reg)*	*reg:* decimal digit 2 - 10.
,DISABLED	**Note:** DISABLED may not be specified if TYPE=CMS or TYPE=LOCAL is specified above.
,REGS=SAVE ,REGS=USE	
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained under the OBTAIN option of the SETLOCK macro instruction, with the following exceptions:

RELEASE
specifies that the lockword is to be released.

,TYPE = SPIN
,TYPE = ALL
,TYPE = *(reg)*
specifies the type of lock that is to be released.

SPIN All spin locks currently held by the CPU are to be released.

ALL All locks currently held by the CPU are to be released.

(reg) The specified register contains a bit string identifying the locks to be released. A value of 1 indicates that the lock held is to be released; a value of 0 indicates that the status of the lock will not change. The bit meanings are:

Bit 19	DISP
Bit 20	ASM
Bit 21	SALLOC
Bit 22	IOSYNCH
Bit 23	IOSCAT
Bit 24	IOSUCB
Bit 25	IOSLCH
Bit 26	Reserved
Bit 27	Reserved
Bit 28	Reserved
Bit 29	SRM
Bit 30	CMS
Bit 31	LOCAL

,DISABLED

specifies that control is to be returned to the caller with the CPU in a physically disabled state (except for machine check) when a lock is successfully released. This form should be used only by those routines which do not have the disabled supervisor indicator on when they are executing and which, upon release of a global spin lock, must remain physically disabled due to noninterruptibility or no recursion restraints.

When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	The lock was successfully released.
04	The lock was not owned. The lock was free when the release request was issued.
08	The release process was unsuccessful. The lockword id does not match the caller's id.

Note: No return codes are supported for multiple releases. That is, return code register contents are unpredictable.

Example 1

Operation: The local lock is requested to be released.

```
SETLOCK RELEASE,TYPE=LOCAL,RELATED=(TCBRQ,MOD1(NAME1),
        MOD2(NAME2))
```

Example 2

Operation: The IOSUCB lock whose address is in register 11 is requested to be released.

```
SETLOCK RELEASE,TYPE=IOSUCB,ADDR=(11),RELATED=(AXYZ,MOD1(LABEL))
```

The TEST option of the SETLOCK macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede SETLOCK.
SETLOCK	
ƀ	One or more blanks must follow SETLOCK.
TEST	
,TYPE=IOSCAT ,TYPE=IOSUCB ,TYPE=IOSLCH ,TYPE=IOSYNCH ,TYPE=ASM ,TYPE=DISP ,TYPE=SALLOC ,TYPE=SRM ,TYPE=CMS ,TYPE=LOCAL ,TYPE=SPIN ,TYPE=ALL ,TYPE=*(reg)*	*reg:* decimal digit 2 - 12

,ADDR=*(reg)*	*reg:* decimal digit 2 - 12
	Note: ADDR may not be specified if any of the following was specified above:
	TYPE=DISP TYPE=LOCAL
	TYPE=SALLOC TYPE=SPIN
	TYPE=SRM TYPE=ALL
	TYPE=CMS TYPE=(reg)

,BRANCH=(HELD,*addr*) ,BRANCH=(NOTHELD,*addr*)	*addr:* RX-type address.
,REGS=*(reg)*	*reg:* decimal digit 2 - 12. **Note:** REGS may only be specified if any of the following was specified above: TYPE=SPIN TYPE=ALL TYPE=(reg)
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained under the OBTAIN or RELEASE option of the SETLOCK macro instruction, with the following exceptions:

TEST

specifies that the lockword is to be checked to determine if it is currently held by the requesting CPU.

,BRANCH = (HELD,*addr*)
,BRANCH = (NOTHELD,*addr*)

specifies that the return code setting output of the macro instruction is to be suppressed and replaced by a direct branch to the specified address.

If HELD is specified, the address will be branched to if the specified lock, or at least one lock for TYPE=ALL or TYPE=SPIN, or all the specified locks for TYPE=*(reg)* are held by the requesting CPU.

If NOTHELD is specified, the address will be branched to if the specified lock is not currently held by the requesting CPU, or if not all the locks specified for TYPE=*(reg)* are held, or if no lock for TYPE=ALL or TYPE=SPIN is held.

,REGS = *(reg)*

specifies the register containing a bit string identifying which locks are held. If the bit string is partially correct (that is, one of the locks specified is not held), the connected string is returned in the register specified.

When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	The lock was held by the requestor, or all the locks were held (if the request was for several locks via a register), or at least one lock was held (if TYPE=ALL or TYPE=SPIN was specified).
04	The lock was not held by anybody, or not all the locks were held (if the request was for several locks via a register), or no lock was held (if TYPE=ALL or TYPE=SPIN was specified).

Example 1

Operation: If the local lock is not held, a branch to DSRLLINT is to be performed; otherwise, the next sequential instruction is to be executed.

```
SETLOCK TEST,TYPE=LOCAL,BRANCH=(NOTHELD,DSRLLINT)
```

SETRP — Set Return Parameters

The SETRP macro instruction is used to indicate the various requests that a recovery exit may return.

The macro instruction is valid only for exits established via functional recovery exits and ESTAE/ESTAI/ESTAR exits. The table following the description of the macro instruction indicates which parameters are valid for each situation.

The SETRP macro instruction is described in the **OS/VS2 Supervisor Services and Macro Instructions**, with the exception of the RECORD, RECPARM, FRELOCK and CPU parameters. These parameters are restricted in use and should be used only by programs excuting in supervisor state or under protection key 0-7 and executing as a functional recovery routine.

The syntax of the complete SETRP macro instruction is shown below. However, only the explanation of the restricted parameters is presented. Explanation of the other parameters can be found in **OS/VS2 Supervisor Services and Macro Instructions**.

Note: This macro instruction requires that the IHASDWA mapping macro be assembled into the caller's csect.

The SETRP macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede SETRP.
SETRP	
ƀ	One or more blanks must follow SETRP.
WKAREA=*(reg)*	*reg:* decimal digits 1-12. **Default:** WKAREA=(1)
,REGS=*(reg1)* ,REGS=*(reg1,reg2)*	*reg1:* decimal digits 0-12, 14, 15. *reg2:* decimal digits 0-12, 14, 15. **Note:** If *reg1* and *reg2* are both specified, order is 14, 15, 0-12.
,DUMP=IGNORE ,DUMP=YES ,DUMP=NO	**Default:** DUMP=IGNORE
,DUMPOPT=*parm list addr*	*parm list addr:* RX-type address, or register (2) - (12). **Note:** This parameter may be specified only if DUMP=YES is specified above.
,RC=0 ,RC=4 ,RC=16	**Default:** RC=0
,RETADDR=*retry addr*	*retry addr:* RX-type address, or register (2) - (12). **Note:** This parameter may be specified only if RC=4 is specified above.
,RETREGS=NO ,RETREGS=YES ,RETREGS=YES,RUB=*info addr*	*info addr:* RX-type address, or register (2) - (12). **Default:** RETREGS=NO **Note:** This parameter may be specified only if RC=4 is specified above.
,FRESDWA=NO ,FRESDWA=YES	**Default:** FRESDWA=NO **Note:** This parameter may be specified only if RC=4 is specified above.
,COMPCOD=*code* ,COMPCOD=(*code*,USER) ,COMPCOD=(*code*,SYSTEM)	*code:* symbol, decimal digit, or register (2) - (12). **Default:** COMPCOD=(*code*,USER)
,FRELOCK=*(locks)*	*locks*: any combination of the following, separated by commas: DISP IOSCAT*(lockword)* SRM IOSUCB*(lockword)* SALLOC IOSLCH*(lockword)* CMS IOSYNCH*(lockword)* LOCAL ASM*(lockword)* *lockword*: RX-type address. **Note:** This parameter may be specified only if RC=0 is specified above.
,CPU=*reg*	*reg:* decimal digits 2-12.
,RECORD=IGNORE ,RECORD=YES ,RECORD=NO	**Default:** RECORD=IGNORE
,RECPARM=*record list addr*	*record list addr:* RX=type address, or register (2) - (12). **Note:** This parameter may be specified only if RECORD=IGNORE or RECORD=YES is specified above.

The parameters restricted in use are explained below. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

,FRELOCK=*(locks)*
specifies the locks to be freed and the corresponding lockwords:
DISP — Global dispatcher lock.
SRM — Systems resource manager lock.
SALLOC — Real storage manager and virtual storage manager space allocation lock.
CMS — Cross memory services lock.
LOCAL — Storage lock of the storage the caller is executing in.
IOSCAT — IOS channel availability table lock.
IOSUCB — IOS unit control block lock.
IOSLCH — IOS logical channel queue lock.
IOSYNCH — IOS synchronization lock.
ASM — Auxiliary storage manager lock.

,CPU=*(reg)*
specifies the register that contains the logical CPU identification of the CPU holding the resource that this CPU is waiting for.

,RECORD=IGNORE
,RECORD=YES
,RECORD=NO
specifies that the entire SDWA, both fixed and variable areas, is to be written on SYS1.LOGREC (YES), is not to be written on SYS1.LOGREC (NO), or is to be written as indicated prior to the SETRP macro instruction (IGNORE).

,RECPARM=*record list addr*
specifies the address of a user-supplied record parameter list. The parameter list consists of three 8-byte fields:

- The first field contains the module name (microfiche name).
- The second field contains the CSECT name.
- The third field contains the FRR identification.

The three fields are left-justified, and padded with blanks.

Note: The variable information record, containing two 2-byte length fields at the beginning of the record consists of:

- The first field, filled in by the system, specifies the total length available to the user (exclusive of the two length fields).
- The second field, filled in by the user, contains the actual length of the record.

The following table indicates which parameters are available to functional recovery routines (FRRs) and which parameters are available to ESTAE/ESTAE exits.

Parameter	**FRR**	**ESTAE**
WKAREA	x	x
REGS	x	x
DUMP	x	x
DUMPOPT	x	x
RC=0	x	x
RC=4	x	x
RC=16		x
RETADDR	x	x
RETREGS	x	x
RUB	x	x
FRESDWA		x
COMPCOD	x	x
FRELOCK	x	
CPU	x	
RECORD	x	x
RECPARM	x	x

Example 1

Operation: Request continue with termination and freeing of the IOSCAT and SRM locks. The IOSCAT lockword is label X.

```
    SETRP    RC=0,FRELOCK=( IOSCAT( X ),SRM )
```

Example 2

Operation: Cause a restart interruption on the CPU identified by the contents of register 7. In this example, the interrupted function is spinning on a lock currently being held by the CPU identified in register 7.

```
    SETRP    CPU=( 7 )
```

SPIE — Specify Program Interruption Exit

The SPIE macro instruction is described in the **OS/VS2 Supervisor Services and Macro Instructions**, with the exception of interruption type 17. This interruption type designates page faults and may be specified by an installation-authorized system programmer.

The syntax of the complete SPIE macro instruction is shown below.

The standard form of the SPIE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede SPIE.
SPIE	
ƀ	One or more blanks must follow SPIE.
exit addr,(interrupts)	*exit addr:* A-type address, or register (2) - (12). *interrupts:* decimal digits 1-15, or 17 expressed as **single values:** (2,3,4,7,8,9,10) **ranges of values:** ((2,4),(7,10)) **combinations:** ((2,4),6,8,(10,13),15)

The parameters are explained below:

exit addr,(interrupts)
specifies the address of the exit routine to be given control when a program interruption of the type specified occurs. The interruption types are:

Number	Interruption Type
1	Operation
2	Privileged operation
3	Execute
4	Protection
5	Addressing
6	Specification
7	Data
8	Fixed-point overflow (maskable)
9	Fixed-point divide
10	Decimal overflow (maskable)
11	Decimal divide
12	Exponent overflow
13	Exponent underflow (maskable)
14	Significance (maskable)
15	Floating-point divide
17	Page fault

Note: If a specified program interruption type is maskable, the corresponding bit is set to 1. Interruption types not specified above are handled by the control program.

Note: As shown in the table above, interruption types can be designated as one or more single numbers, as one or more pairs of numbers (designating ranges of values), or as any combination of the two forms. For example, (4,8) indicates interruption types 4 and 8; ((4,8)) indicates interruption types 4 through 8.

Example 1

Operation: Give control to an exit routine for interruptions 1,5,7,8,9, and 10. DOITSPIE is the address of the SPIE exit routine.

```
SPIE    DOITSPIE,(1,5,7,(8,10))
```

SPIE (List Form)

Use the list form of the SPIE macro instruction to construct a control program parameter list in the form of a program interruption control area.

The list form of the SPIE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede SPIE.
SPIE	
ƀ	One or more blanks must follow SPIE.
exit addr	*exit addr:* A-type address.
,(interrupts)	*interrupts:* decimal digits 1-15, or 17, expressed as **single values:** (2,3,4,7,8,9,10) **ranges of values:** ((2,4,),(7,10)) **combinations:** ((2,4),6,8,(10,13),15)
,MF=L	

The parameters are explained under the standard form of the SPIE macro instruction, with the following exceptions:

,MF=L
specifies the list form of the SPIE macro instruction.

SPIE (Execute Form)

A remote control program parameter list (program interruptions control area) is used in, and can be modified by, the execute form of the SPIE macro instruction. The PICA (program interruptions control area) can be generated by the list form of SPIE, or you can use the address of the PICA returned in register 1 following a previous SPIE macro instruction. If this macro instruction is being issued to reestablish a previous SPIE environment, code only the MF parameter.

The address of the remote control program parameter list associated with any previous SPIE environment is returned by the SPIE macro instruction.

The execute form of the SPIE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede SPIE.
SPIE	
ƀ	One or more blanks must follow SPIE.
exit addr	*exit addr:* RX-type address, or register (2) - (12).
,*(interrupts)*	*interrupts:* decimal digits 1-15, or 17, expressed as **single values:** (2,3,4,7,8,9,10) **ranges of values:** ((2,3),(7,10)) **combinations:** ((2,4),6,8,(10,13),15)
,MF=(E,*ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

The parameters are explained under the standard form of the SPIE macro instruction, with the following exceptions:

,MF = *(E,ctrl,addr)*
specifies the execute form of the SPIE macro instruction using a remote control program parameter list.

SPOST — Synchronize POST

The SPOST macro instruction is used in a cross-memory post environment to ensure that all outstanding cross-memory post requests for the ECB specified have completed. SPOST resolves a synchronization problem that arises when it becomes necessary to eliminate an ECB which is a potential target for a cross-memory post request.

For explanation of the parameters in a cross-memory post request, see the POST macro instruction.

SPOST invokes the PURGEDQ SVC. For details, see the PURGEDQ macro instruction.

The SPOST macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede SPOST.
SPOST	

Note: SPOST contains no optional or required parameters.

Example 1

Operation: Execute the SPOST macro instruction, with a comment.

```
SPOST    ,ISSUE SPOST
```

STAE — Specify Task Abnormal Exit

The STAE macro instruction enables the user to intercept a scheduled ABEND and to have control returned to him at a specified exit routine address. The STAE macro instruction operates in both problem program and supervisor modes.

Note: The STAE macro instruction is available for compatibility with release 1 of VS2 and with MFT and MVT. However, it is recommended that ESTAE be used.

The standard form of the STAE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede STAE.
STAE	
ƀ	One or more blanks must follow STAE.
exit addr 0	*exit addr:* A-type address, or register (2) - (12).
,CT ,OV	**Default:** CT
,PARAM=*list addr*	*list addr:* A-type address, or register (2) - (12).
,XCTL=NO ,XCTL=YES	**Default:** XCTL=NO
,PURGE=QUIESCE ,PURGE=HALT ,PURGE=NONE	**Default:** PURGE=QUIESCE
,ASYNCH=NO ,ASYNCH=YES	**Default:** ASYNCH=NO
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained below:

exit addr
0
specifies the address of a STAE exit routine to be entered if the task issuing this macro instruction terminates abnormally. If 0 is specified, the most recent STAE request is canceled.

,CT
,OV
specifies the creation of a new STAE exit (CT) or indicates that the parameters passed in this STAE macro instruction are to overlay the data contained in the previous STAE exit (OV).

,PARAM = *list addr*
specifies the address of a user-defined parameter list containing data to be used by the STAE exit routine when it is scheduled for execution.

,XCTL = NO
,XCTL = YES
specifies that the STAE macro instruction will be canceled (NO) or will not be canceled (YES) if an XCTL macro instruction is issued by this program.

,PURGE=QUIESCE
,PURGE=HALT
,PURGE=NONE

specifies that all outstanding requests for I/O operations will not be saved when the STAE exit is taken (HALT), that I/O processing will be allowed to continue normally when the STAE exit is taken (NONE), or that all outstanding requests for I/O operations will be saved when the STAE exit is taken (QUIESCE). For QUIESCE, at the end of the STAE exit routine, the user can code a retry routine to handled the outstanding I/O requests.

Note: If any IBM-supplied access method, except EXCP, is being used, the PURGE=NONE option is recommended. If this is done, all control blocks affected by input/output processing may continue to change during STAE exit routine processing.

If PURGE=NONE is specified and the ABEND was originally scheduled because of an error in input/output processing, an ABEND recursion will develop when an input/output interruption occurs, even if the exit routine is in progress. Thus, it will appear that the exit routine failed when, in reality, input/output processing was the cause of the failure.

ISAM Notes: If ISAM is being used and PURGE=HALT is specified or PURGE=QUIESCE is specified but I/O is not restored:

- Only the input/output event on which the purge is done will be posted. Subsequent event control blocks (ECBs) will not be posted.
- The ISAM check routine will treat purged I/O as normal I/O.
- Part of the data set may be destroyed if the data set is being updated or added to when the failure occurred.

,ASYNCH=NO
,ASYNCH=YES

specifies that asynchronous exit processing will be allowed (YES) or will not be allowed (NO) while the STAE exit is executing.

ASYNCH=YES must be coded if:

- Any supervisor services that require asynchronous interruptions to complete their normal processing are going to be requested by the STAE exit routine.
- PURGE=QUIESCE is specified for any access method that requires asynchronous interruptions to complete normal input/output processing.
- PURGE=NONE is specified and the CHECK macro instruction is issued in the STAE exit routine for any access method that requires asynchronous interruptions to complete normal input/output processing.

Note: If ASYNCH=YES is specified and the ABEND was originally scheduled because of an error in asynchronous exit handling, an ABEND recursion will develop when an asynchronous interruption occurs. Thus, it will appear that the exit routine failed when, in reality, asynchronous exit handling was the cause of the failure.

,RELATED=*value*

specifies information used to self-document macro instructions by 'relating' functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

The RELATED parameter is available on macro instructions that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE, and on macro instructions that relate to previous occurrences of the same macro instructions (for example, CHAP and ESTAE).

The parameter may be used, for example, as follows:

```
GET1     GETMAIN     R,LV=4096,RELATED=(FREE1,'GET STORAGE')
FREE1    FREEMAIN    R,LV=4096,A=(1),RELATED=(GET1,'FREE STORAGE')
```

Control is returned to the instruction following the STAE macro instruction. When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful completion of STAE request.
04	STAE was unable to obtain storage for STAE request.
08	Attempt was made to cancel or overlay a nonexistent STAE request.
0C	Exit routine or parameter list address was invalid, or STAI request was missing a TCB address.
10	Attempt was made to cancel or overlay a STAE request of another user, or an unexpected error was encountered while processing this request.

Example 1

Operation: Request an overlay of the existing STAE recovery exit with the following options: new exit address is ADDR, parameter list is at PLIST, I/O will be halted, no asynchronous exits will be taken, ownership will be transferred to the new request block resulting from any XCTL macro instructions.

```
STAE     ADDR,OV,PARAM=PLIST,XCTL=YES,PURGE=HALT,ASYNCH=NO
```

STAE (List Form)

The list form of the STAE macro instruction is used to construct a remote control program parameter list.

The list form of the STAE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede STAE.
STAE	
ƀ	One or more blanks must follow STAE.
exit addr	*exit addr:* A-type address.
,PARAM=*list addr*	*list addr:* A-type address.
,PURGE=QUIESCE ,PURGE=HALT ,PURGE=NONE	**Default:** PURGE=QUIESCE
,ASYNCH=NO ,ASYNCH=YES	**Default:** ASYNCH=NO
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

The parameters are explained under the standard form of the STAE macro instruction, with the following exceptions:

,MF=L
specifies the list form of the STAE macro instruction.

STAE (Execute Form)

A remote control program parameter list is used in, and can be modified by, the execute form of the STAE macro instruction. The control program parameter list can be generated by the list form of the STAE macro instruction. If the user desires to dynamically changed the contents of the remote STAE parameter list, he may do so by coding a new exit address and/or a new parameter list address. If exit address or PARM= is coded, only the associated field in the remote STAE parameter list will be changed. The other field will remain as it was before the current STAE request was made.

The execute form of the STAE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede STAE.
STAE	
ƀ	One or more blanks must follow STAE.
exit addr 0	*exit addr:* RX-type address, or register (2) - (12).
,CT ;OV	
,PARAM=*list addr*	*list addr:* RX-type address, or register (2) - (12).
,XCTL=NO ,XCTL=YES	
,PURGE=QUIESCE ,PURGE=HALT ,PURGE=NONE	
,ASYNCH=NO ,ASYNCH=YES	
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=(E, *ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

The parameters are explained under the standard form of the STAE macro instruction, with the following exceptions:

,MF = (E, *ctrl addr*)
specifies the execute form of the STAE macro instruction using a remote control program parameter list.

Example 1

Operation: Provide the pointer to the recovery code in the register called EXITPTR, contain the address of the STAE exit parameter list in register 9. Register 8 points to the area where the STAE parameter list (created with the MF=L option) was moved.

```
STAE    (EXITPTR),PARAM=(9),MF=(E,(8))
```

STATUS — Change Subtask Status

The STATUS macro instruction is described in the **OS/VS2 Supervisor Services and Macro Instructions,** with the exception of those parameters that are restricted in use and available only to authorized callers. These restricted parameters allow the caller to manipulated the dispatchability of TCBs, SRBs, ASCBs, a STEP, or the SYSTEM.

The description of the STATUS macro instruction has been divided into two parts: the START/STOP option, and the SET/RESET option.

The START/STOP options of the STATUS macro instruction are written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede STATUS.
STATUS	
ƀ	One or more blanks must follow STATUS.
START STOP	
,TCB=*tcb addr* ,SRB ,SRB,ASID=*ASID addr* ,SYNCH	*tcb addr:* RX-type address, or register (2) - (12), or 0. *ASID addr:* RX-type address, or register (2) - (12). **Note:** ASID may only be specified with START. **Note:** SYNCH may only be specified with STOP.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained below:

START
STOP
specifies that the START/STOP count is to be adjusted and the dispatchability bits are to be set/reset.

,TCB=*tcb addr*
,SRB
,SRB,ASID=*ASID addr*
,SYNCH
specifies the status of the stop/start function:

TCB specifies the address of a fullword on a fullword boundary containing the address of the TCB that is to have its START/STOP count adjusted.

SRB specifies that the STOP function affects the dispatchability of system-level SRBs only; all other tasks in the address space area set/reset nondispatchable. For START, the ASID addr specifies the address of a halfword containing the address space identifier.

SYNCH specifies that the STOP function affects all the subtasks of the caller. If any of the subtasks are deferring STOPs when the request is issued, the caller is placed in a WAIT condition. The issuer is taken out of the wait state when all deferred stops are complete.

,RELATED=*value*
specifies information used to self-document macro instructions by 'relating' functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

The RELATED parameter is available on macro instructions that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE, and on macro instructions that relate to previous occurrences of the same macro instructions (for example, CHAP and ESTAE).

The parameter may be used, for example, as follows:

```
GET1     GETMAIN     R,LV=4096,RELATED=(FREE1,'GET STORAGE')
FREE1    FREEMAIN    R,LV=4096,A=(1),RELATED=(GET1,'FREE STORAGE')
```

The SET/RESET options of the STATUS macro instruction are written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede STATUS.
STATUS	
ƀ	One or more blanks must follow STATUS.
SET RESET	
,MC ,MC,STEP ,SD ,ND	**Note:** If MC or MC,STEP is specified, no other parameters may be specified.
,SYSTEM ,STEP ,STEP,*(mask)* *,tcb addr,(mask)* *,,(mask)*	*mask:* for SD, any of decimal digits 1-32 (except 18), separated by commas; for ND, any of decimal digits 1-16 (except 14), separated by commas. *tcb addr:* RX-type address, or register (2) - (12). **Default:** STEP
,E	**Note:** This parameter may only be specified with *tcb addr,(mask).*
,ASID=*ASID addr*	*ASID addr:* RX-type address, or register (2) - (12). **Note:** For SET, this parameter may only be specified with *tcb addr,(mask).* For RESET, this parameter may *not* be specified with SYSTEM.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained below:

SET
RESET
specifies that the TCBs or ASCBs are to be set or reset nondispatchable.

,MC
,MC,STEP
,SD
,ND
specifies the nondispatchability status:

ND specifies that the primary nondispatchability bits are affected by this request.

SD specifies that the secondary nondispatchability bits are affected by this request.

MC and MC,STEP specifies that the initiator and all TCBs in the job step TCBs (except the issuer's TCB) are to be set/reset nondispatchable. STEP indicates that the set-must-complete indicator in the issuer's TCB and a count in the ASCB are to be set/reset.

,SYSTEM
,STEP
,STEP,*(mask)*
,tcb addr,(mask)
,,(mask)
specifies more information on the nondispatchability status:

SYSTEM specifies that all ASCBs are to be set/reset nondispatchable except for certain exempt ones (for examples, the master scheduler or the issuer).

STEP specifies that all job step TCBs (except the issuer's TCB) are to be set/reset nondispatchable.

tcb addr specifies that the specified TCB and all its subtasks are to be set/reset nondispatchable.

(mask) specifies the nondispatchability bits that are to be set/reset.

,E
specifies that only the specified TCB is to be set/reset nondispatchable.

,ASID=*ASID addr*
specifies the address of a halfword containing the address space identifier.

,RELATED=*value*
specifies information used to self-document macro instructions by 'relating' functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

The RELATED parameter is available on macro instructions that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE, and on macro instructions that relate to previous occurrences of the same macro instructions (for example, CHAP and ESTAE).

The parameter may be used, for example, as follows:

```
GET1     GETMAIN     R,LV=4096,RELATED=(FREE1,'GET STORAGE')
FREE1    FREEMAIN    R,LV=4096,A=(1),RELATED=(GET1,'FREE STORAGE')
```

Example 1

Operation: Set primary nondispatchability bit 3 for the specified TCB and all its subtasks.

```
STATUS   SET,ND,TCBADDR,(3)
```

SYNCH — Take a Synchronous Exit to a Processing Program

The SYNCH macro instruction makes it possible for a supervisor routine to take a synchronous exit to a processing program. The SYNCH routine initializes a PRB (program request block) and schedules execution of the requested program. After the processing program has been executed, the supervisor routine that issued the SYNCH macro instruction regains control.

The SYNCH macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede SYNCH.
SYNCH	
ƀ	One or more blanks must follow SYNCH.
entry name addr	*entry name addr:* RX-type address, or register (2) - (12) or (15).

The parameters are explained below:

entry name addr
specifies the address of the entry name of the processing program to receive control.

Example 1

Operation: Take a synchronous exit to a processing program whose entry name address is specified in register 8.

```
SYNCH   (8)
```

Example 2

Operation: Take a synchronous exit to a processing program labeled SUBRTN.

```
SYNCH   SUBRTN
```

TESTAUTH — Test Authorization of Caller

The TESTAUTH macro instruction is used on behalf of a privileged or sensitive function to verify that its caller is appropriately authorized.

TESTAUTH supports the authorized program facility (APF) - a facility that permits the identification of programs that are authorized to use restricted functions. In addition, TESTAUTH provides the capability for testing for system key 0-7 and supervisor state.

The TESTAUTH macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede TESTAUTH.
TESTAUTH	
ƀ	One or more blanks must follow TESTAUTH.

FCTN=*fctn*
FCTN=*fctn*,AUTH=*auth*
	fctn: decimal digit 1 or register (2) - (12).
	auth: decimal digit 0 or 1, or register (2) - (12).

,STATE=NO ,STATE=YES	Default: STATE=NO
,KEY=NO ,KEY=YES	Default: KEY=NO
,RBLEVEL=2 ,RBLEVEL=1	Default: RBLEVEL=2
,BRANCH=NO ,BRANCH=YES	Default: BRANCH=NO

The parameters are explained below:

FCTN=*fctn*
FCTN=*fctn*,AUTH=*auth*
specifies the authorization (via APF) of a program.

FCTN=1 specifies that APF-authorization is checked.

AUTH=0 specifies that the job step is not authorized to perform any restricted function.

AUTH=1 specifies that the job step is authorized to perform restricted functions.

Note: If FUNC=1 is specified by itself (that is, without the AUTH parameter), the JSCB is used to check for authorization. AUTH should only be coded when it is not possible for TESTAUTH to acquire the code from the JSCB.

,STATE=NO
,STATE=YES
specifies whether or not (YES or NO) a check is to be made for supervisor/problem program state. (Supervisor state is authorized, problem program state is not authorized.)

,KEY=NO
,KEY=YES
specifies whether or not (YES or NO) a check is to be made of the protection keys. (Protection keys 0-7 are authorized, protection keys 8-15 are not authorized.)

,RBLEVEL=2
,RBLEVEL=1
specifies whether the TESTAUTH caller is a type 2, 3, or 4 SVC (RBLEVEL=2), or a type 1 SVC (RBLEVEL=1).

,BRANCH = NO
,BRANCH = YES
specifies a branch entry (YES) or an SVC entry (NO).

When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	**Meaning**
00	Task is authorized.
04	Task is not authorized.

Example 1

Operation: Test jobstep for APF authorization.

```
TESTAUTH     AUTH=1,FCTN=1
```

Example 2

Operation: Test for APF authorization and supervisor state, and do not check protection keys.

```
TESTAUTH     STATE=YES,KEY=NO,FCTN=1
```

WTO — Write to Operator

The WTO macro instruction is described in the **OS/VS2 Supervisor Services and Macro Instructions**, with the exception of the AREAID, MSGTYP, and MCSFLAG parameters. The MSGTYP and MCSFLAG parameters should only be used by system programmers familiar with MCS, since using the parameters improperly could impede the entire message routing scheme. The AREAID parameter can only be be used by APF-authorized users.

The syntax of the complete WTO macro instruction is shown below. However, only the explanation of the AREAID, MSGTYP and MCSFLAG parameters are presented. Explanation of the other parameters can be found in **OS/VS2 Supervisor Services and Macro Instructions.**

The standard form of the WTO macro instruction is written as follows:

name	*name:* symbol. Begin name in column 1.
ƀ	One or more blanks must precede WTO.
WTO	
ƀ	One or more blanks must follow WTO.
'msg' *('text')* *('text',line type)*	*msg:* Up to 124 characters. *text:* Up to 124 characters. The permissable *line types* and *text* lengths are shown below: line type VS2 text C 34 char L 70 char D 70 char DE 70 char E ------ Up to 10 occurrences of the second and/or third formats ('text' or 'text', line type) may be coded.
,ROUTCDE=*(route code)*	*route code:* decimal digit from 1 to 16. The *route code* is one or more codes, separated by commas.
,DESC=*(desc code)*	*desc code:* decimal digit from 1 to 16. The *desc code* is one or more codes, separated by commas.
,AREAID=*id char*	*id char:* alphabetic character A - Z.
,MSGTYP=*(msg type)*	*msg type:* any of the following N SESS,JOBNAMES Y SESS,STATUS SESS JOBNAMES,STATUS JOBNAMES SESS,JOBNAMES,STATUS STATUS
,MCSFLAG=*(field name)*	*field name:* any combination of the following, separated by commas: REG0 HRDCPY RESP QREG0 REPLY NOTIME BRDCST NOCPY

The parameters restricted in use are explained below. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instruction.**

,AREAID = *id char*

specifies a display area of the console screen on which a multiple-line message is to be written. This parameter is useful only for out-of-line (descriptor code 8 and 9) MLWTO messages which are be sent to CRT consoles.

The character Z designates the message area (the screen's general message area, rather than a defined display area); it is assumed nothing is specified.

Note: You must be APF-authorized to use this parameter. Also, if an area is specified by this parameter, there exists the possibility that this area will be overlaid by a currently running dynamic display.

,MSGTYP = *(msg type)*

specifies how the message is to be routed.

For SESS, JOBNAMES, or STATUS, the message is to be routed to the console and TSO terminal in operator mode which issued the MONITOR SESS, MONITOR JOBNAMES, or MONITOR STATUS command, respectively. When the message type is identified by the operating system, the message will be routed to only those consoles that had requested the information.

For Y or N, the message type specifies whether flags are to be set in the WTO macro expansion to describe what functions (MONITOR SESS, MONITOR JOBNAMES, and MONITOR STATUS) are desired. N, or omission of the MSGTYP parameter, indicates that the message is to be routed as specified in the ROUTCDE parameter.

,MCSFLAG = *(field name)*

specifies that the macro expansion should set bits in the MCSFLAG field as indicated by each name coded. The names and corresponding bit settings are shown in Figure 30.

Name	Bit	Meaning
----	0	Invalid entry.
REG0	1	Message is to be queued to the console whose scource ID is passed in register 0.
RESP	2	The WTO is an immediate command response.
----	3	Invalid entry.
REPLY	4	The WTO macro instruction is a reply to a WTOR macro instruction.
BRDCST	5	Message should be broadcast to all active consoles.
HRDCPY	6	Message queued for hard copy only.
QREG0	7	Message is to be queued unconditionally to the console whose source ID is passed in Register 0.
NOTIME	8	Time is not appended to the message.
---	9-12	Invalid entry.
NOCPY	13	If the WTO or WTOR macro instruction is issued by a program in the supervisor state, the message is not queued for hard copy. Otherwise, this parameter is ignored.
---	14-15	Invalid entry.

Note: Invalid specifications are ignored and produce an appropriate error message from the assembler.

Figure 30. MCSFLAG Fields

Example 1

Operation: Send a WTO message to the hardcopy log only.

```
WTO     'THIS MSG IS TO HARDCOPY ONLY WITH RC=ALL',MCSFLAG=HRDCPY,
        ROUTCDE=(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16)
```

Example 2

Operation: Send a WTO message to all active consoles and broadcast it to all consoles or terminals which have issued MONITOR commands.

```
WTO      'THIS MSG IS BROADCAST WITH RC=ALL',MCSFLAG=BRDCST,
         ROUTCDE=(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16)
```

Example 3

Operation: Send a WTO message to all consoles and TSO terminals which have issued a MN JOBNAMES command.

```
WTO      'WTO BY MSGTYP=JOBNAMES WITH RC=ALL,NO CONSOLE MONITORING
         JOBNAMES',MSGTYP=JOBNAMES,ROUTCDE=(1,2,3,4,5,6,7,8,9,10,11,
         12,13,14,15,16)
```

WTO (List Form)

The list form of the WTO macro instruction is described in the **OS/VS2 Supervisor Services and Macro Instructions**, with the exception of the AREAID, MSGTYP and MCSFLAG parameters. These parameters are restricted in use, and are described below.

The list form of the WTO macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede WTO.
WTO	
ƀ	One or more blanks must follow WTO.
'msg' *('text')* *('text',line type)*	*msg:* up to 124 characters. The permissable *line types* and *text* lengths are shown below: line type VS2 text C 34 char L 70 char D 70 char DE 70 char E ------ Up to 10 occurrences of the second and/or third formats ('text' or 'text', line type) may be coded.
,ROUTCDE=*(route code)*	*route code:* decimal digit from 1 to 16. The *route code* is one or more codes, separated by commas.
,DESC=*(desc code)*	*desc code:* decimal digit from 1 to 16. The *desc code* is one or more codes, separated by commas.
,AREAID=*id char*	*id char:* an alphabetic character A-Z.
,MSGTYP=*(msg type)*	*msg type:* any one of the following: N SESS,JOBNAMES Y SESS,STATUS SESS JOBNAMES,STATUS JOBNAMES SESS,JOBNAMES,STATUS STATUS
,MCSFLAG=*(field name)*	*field name:* any combination of the following, separated by commas: REG0 HRDCPY RESP QREG0 REPLY NOTIME BRDCST NOCPY
,MF=L	

The parameters restricted in use are explained under the standard form of the WTO macro instruction. The explanation of the other parameters is as explained in **OS/VS2 Supervisor Services and Macro Instructions.**

WTOR — Write to Operator with Reply

The WTOR macro instruction is described in the **OS/VS2 Supervisor Services and Macro Instructions**, with the exception of the MSGTYP and MCSFLAG parameters. These parameters should only be used by system programmers familiar with MCS, since using the parameters improperly could impede the entire message routing scheme.

The syntax of the complete WTOR macro instruction is shown below. However, only the explanation of the MSGTYP and MCSFLAG parameters are presented. Explanation of the other parameters can be found in **OS/VS2 Supervisor Services and Macro Instructions.**

The standard form of the WTOR macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede WTOR.
WTOR	
ƀ	One or more blanks must follow WTOR.
msg	*msg:* up to 121 characters.
,reply addr	*reply addr:* A-type address, or register (2) - (12).
,reply length	*reply length:* symbol, decimal digit, or register (2) - (12). The minimum length is 1; the maximum length is 115 when the operator enters REPLY id, 'reply' and 119 when the operator enters R id, 'reply'.
,ecb addr	*ecb addr:* A-type address, or register (2) - (12).
,ROUTCDE=*(route code)*	*route code:* decimal digit from 1 to 16. The *route code* is one or more codes, separated by commas.
,DESC=*(desc code)*	*desc code:* decimal digit from 1 to 16. The *desc code* is one or more codes, separated by commas.
,MSGTYP=*(msg type)*	*msg type:* any one of the following: N SESS,JOBNAMES Y SESS,STATUS SESS JOBNAMES,STATUS JOBNAMES SESS,JOBNAMES,STATUS STATUS
,MCSFLAG=*(field name)*	*field name:* any combination of the following, separated by commas: REG0 HRDCPY RESP QREG0 REPLY NOTIME BRDCST NOCPY

The parameters restricted in use are explained below. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions.**

,MSGTYP=*(msg type)*
specifies how the message is to be routed.

For SESS, JOBNAMES, or STATUS, the message is to be routed to the console or TSO terminal in operator mode which issued the MONITOR SESS, MONITOR JOBNAMES, or MONITOR STATUS command, respectively. When the message type is identified by the operating system, the message will be routed to only those consoles that had requested the information.

For Y or N, the message type specifies whether flags are to be set in the WTO macro expansion to describe what functions (MONITOR SESS, MONITOR JOBNAMES, and

MONITOR STATUS) are desired. N, or omission of the MSGTYP parameter, indicates that the message is to be routed as specified in the ROUTCDE parameter.

,MCSFLAG = *(field name)*
specifies that the macro expansion should set bits in the MCSFLAG field as indicated by each name coded. The names and corresponding bit settings are shown in Figure 24 that appears in the description of WTO.

Example 1

Operation: Send a WTOR message to the hardcopy log only.

```
WTOR     'THIS MSG IS TO HARDCOPY ONLY WITH RC=ALL',MCSFLAG=HRDCPY,
         ROUTCDE=(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16)
```

Example 2

Operation: Send a WTOR message to all active consoles and broadcast it to all consoles or terminals which have issued MONITOR commands.

```
WTOR     'THIS MSG IS BROADCAST WITH RC=ALL',MCSFLAG=BRDCST,
         ROUTCDE=(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16)
```

Example 3

Operation: Send a WTOR message to all consoles and TSO terminals which have issued a MN JOBNAMES command.

```
WTOR     'WTOR BY MSGTYP=JOBNAMES WITH RC=ALL, NO CONSOLE MONITORING
         JOBNAMES',MSGTYP=JOBNAMES,ROUTCDE=(1,2,3,4,5,6,7,8,9,10,11,
         12,13,14,15,16)
```

WTOR (List Form)

The list form of the WTOR macro instruction is described in the **OS/VS2 Supervisor Services and Macro Instructions**, with the exception of the MSGTYP and MCSFLAG parameters. These parameters are restricted in use, and are described below.

The list form of the WTOR macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede WTOR.
WTOR	
ƀ	One or more blanks must follow WTOR.
'msg'	*msg:* up to 121 characters.
, *,reply addr*	*reply addr:* an A-type address.
, *,reply length*	*reply length:* symbol or decimal digit. The minimum length is 1; the maximum length is 115 when the operator enters REPLY id, 'reply' and 119 when the operator enters R id, 'reply'.
, *,ecb addr*	*ecb addr:* A-type address.
,ROUTCDE=*(route code)*	*route code:* decimal digit from 1 to 16. The *route code* is one or more codes, separated by commas.
,DESC=*(desc code)*	*desc code:* decimal digit from 1 to 16. The *desc code* is one or more codes, separated by commas.
,MSGTYP=*(msg type)*	*msg type:* any one of the following: N SESS,JOBNAMES Y SESS,STATUS SESS JOBNAMES,STATUS JOBNAMES SESS,JOBNAMES,STATUS STATUS
,MCSFLAG=*(field name)*	*field name:* any combination of the following, separated by commas: REG0 HRDCPY RESP QREG0 REPLY NOTIME BRDCST NOCPY
,MF=L	

The parameters restricted in use are explained under the standard form of the WTOR macro instruction. The other parameters are explained in **OS/VS2 Supervisor Services and Macro Instructions**.

Index

GC28-0628-1

OS/VS2 SPL: Supervisor (S370-36) Printed in U.S.A. GC28-0628-1

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

OS/VS2 System Programming Library:
Supervisor
GC28-0628-1

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Index Figures Examples Legibility

Are the tables used to describe the macro instructions in this publication an improvement over the brackets-and-braces syntax?

What is your occupation? ______________________________

Number of latest Technical Newsletter (if any) concerning this publication: ______________

Please indicate your address in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Cut or Fold Along Line

GC28-0628-1

Your comments, please . . .

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Cut or Fold Along Line

Fold   Fold

First Class
Permit 81
Poughkeepsie
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department D58, Building 706-2
PO Box 390
Poughkeepsie, New York 12602

Fold   Fold

OS/VS2 SPL: Supervisor (S370-36)   Printed in U.S.A.   GC28-0628-1

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

IBM **Technical Newsletter**

This Newsletter No.	GN28-2634
Date	January 16, 1976
Base Publication No.	GC28-0628-1
File No.	S370-36
Previous Newsletters	None

OS/VS2 System Programming Library: Supervisor

This Technical Newsletter, a part of release 3.7 of OS/VS2, provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent releases unless specifically altered. Pages to be inserted and/or removed are:

Cover, Edition Notice
9, 10
177 - 180

A change to the text or to an illustration is indicated by a vertical line to the left of the change. Some pages without changes were reprinted in order to decrease the number of page inserts.

Summary of Amendments

The CLEAR keyword parameter has been added to the SETFRR macro instruction.

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Publications Development, Department D58, Building 706-2, PO Box 390, Poughkeepsie, New York 12602

IBM Technical Newsletter

This Newsletter	GN28-2740
Date	June 15, 1976
Base Publication No.	GC28-0628-1
File No.	S370-36
Previous Newsletters	GN28-2634

OS/VS2 System Programming Library: Supervisor

This Technical Newsletter, a part of release 3.7 of OS/VS2, provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent releases unless specifically altered. Pages to be inserted and/or removed are:

Cover, Edition Notice	
9, 10	
89, 90	(includes 90.1, 90.2)

A change to the text or to an illustration is indicated by a vertical line to the left of the change.

Summary of Amendments

Changes have been made to reflect APAR fixes to Release 3.7.

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Publications Development, Department D58, Building 706-2, PO Box 390, Poughkeepsie, New York 12602