GC28-1151-4
File No. S370-36

MVS/Extended Architecture System Programming Library: System Macros and Facilities Volume 2

Program Product

MVS/System Product:

JES3 Version 2	**5665-291**
JES2 Version 2	**5740-XC6**

Fifth Edition (June, 1987)

This is a major revision of, and obsoletes, GC28-1151-3. See the Summary of Amendments following the Contents for a summary of the changes made to this manual. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

This edition, with Technical Newsletter GN28-1096, applies to Version 2 Release 2, and all subsequent releases of MVS/System Product 5665-291 or 5740-XC6 until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370 Bibliography*, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products or services do not state or imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product in this publication is not intended to state or imply that only IBM's product may be used. Any functionally equivalent product may be used instead.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Department D58, Building 921-2, PO Box 390, Poughkeepsie, N.Y. 12602. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

This two-volume publication describes supervisor and scheduler facilities that the system programmer can use. In this publication, a system programmer is defined as a programmer whose programs run in supervisor state, system key 0-7 or access APF-authorized libraries. The publication includes the macro instructions and parameters used to obtain the functions.

Volume 1, GC28-1150, contains descriptions of the supervisor and scheduler services available to a system programmer. Most of the services described are supervisor services; however, the scheduler functions available through the use of the DYNALLOC macro instruction are also described. Volume 1 includes a description of the DYNALLOC macro instruction. Some of the topics discussed in Volume 1 are also discussed in *Supervisor Services and Macro Instructions*; however in Volume 1, these topics are extended to include functions that are restricted to system programmers or used primarily by system programmers.

Volume 2 contains the formats and descriptions of the supervisor macro instructions. Volume 2 provides system programmers with the information necessary to code the macro instructions. Each macro instruction is completely described, in Volume 2, but restrictions, requirements, and environmental considerations for the effective use of each macro is explained in Volume 1.

Publications referenced:

Assembler H Version 2 Application Programming: Language Reference, GC26-4037

MVS/Extended Architecture Debugging Handbook Volume 1, LC28-1164

MVS/Extended Architecture Debugging Handbook Volume 2, LC28-1165

MVS/Extended Architecture Debugging Handbook Volume 3, LC28-1166

MVS/Extended Architecture Debugging Handbook Volume 4, LC28-1167

MVS/Extended Architecture Debugging Handbook Volume 5, LC28-1168

MVS/Extended Architecture Interactive Problem Control System User's Guide and Reference, GC28-1297

MVS/Extended Architecture Interactive Problem Control System Logic and Diagnosis, GC28-1298

MVS/Extended Architecture Message Library: System Codes, GC28-1157

OS/VS2 Planning: Global Resource Serialization, GC28-1062

MVS/Extended Architecture Supervisor Services and Macro Instructions, GC28-1154

MVS/Extended Architecture System Logic Library Volume 12, LY28-1250

MVS/Extended Architecture System Programming Library: Initialization and Tuning, GC28-1149

MVS/Extended Architecture System Programming Library: Service Aids, GC28-1159

System Programming Library: Resource Access Control Facility (RACF), SC28-1343

370-Extended Architecture: Principles of Operation, GA22-7085

Notes:

1. *All references to RACF in this publication indicate the program product Resource Access Control Facility Version 1 Release 7 (5740-XXH).*
2. *All references to Assembler H in this publication indicate the program product Assembler H Version 2 (5668-962).*

Contents

Figures

Summary of Amendments

Summary of Amendments
for GC28-1151-4
MVS/System Product Version 2 Release 2

This major revision describes the new BLSQSHDR, IEFQMREQ, IOSDDT, IOSDMLT, SWAREQ, and SYMREC macros, and changes in the BLSQMFLD, DOM, SDUMP, SETRP, SVCUPDTE, VSMLOC, WTO, and WTOR, macros. It also describes changes that affect:

- The DATOFF index entry, INDCDS.
- The GSPV and GSPL parameters of the ATTACH macro.

Summary of Amendments
for GC28-1151-3
for the following:
- MVS/System Product Version 2
Release 1.3 Vector Facility Enhancement
- MVS/System Product Version 2
Release 1.3 Availability Enhancement
- RACF Version 1 Release 7

In support of MVS/System Product Version 2, Release 1.3 Vector Facility Enhancement, this revision contains changes to the BLSQMFLD, CALLDISP, ESPIE, SNAP, and SPIE macro instructions for the Vector Facility.

In support of both MVS/System Product Version 2 Release 1.3 Availability Enhancement and MVS/System Product Version 2 Release 1.3 Vector Facility Enhancement, this revision contains changes to the CALLDISP macro instruction.

In support of RACF Version 1 Release 7, this revision contains changes to the RACDEF, RACHECK, RACINIT, RACLIST, RACROUTE, and RACXTRT macro instructions.

This revision also contains minor technical and editorial updates.

Summary of Amendments
for GC28-1151-2
for MVS/System Product Version 2 Release 1.3

This revision documents the new IOSINFO macro in support of MVS/System Product Version 2, Release 1.3 and maintenance changes to numerous other macros.

Using the Supervisor Macro Instructions

You can communicate service requests to the control program using a set of macro instructions provided by IBM. The users of most of the macro instructions described in this publication must be in supervisor state or PSW key 0-7 or APF-authorized or PKM 0-7; that is, MVS restricts their use. MVS does not restrict some of the macro instructions described in this publication, but because of the functions of the macro instructions, the installation might want to restrict them.

This volume describes those supervisor macro instructions that should be installation-controlled. The supervisor macro instructions intended for the application programmer are described in *Supervisor Services and Macro Instructions.* Some macro instructions are totally restricted in use; others are not restricted in use, but do contain some restricted parameters. For each macro instruction, any restrictions are described first, followed by the macro syntax and a complete description.

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, a supervisor call (SVC), or a PC instruction to give control to the proper program. The exact macro expansion appears as part of the assembler output listing.

Selecting the Macro Level

Certain MVS/XA macro expansions cannot execute on an MVS/370 system. These macros are downward incompatible. Parameters that are new for MVS/XA are not supported by the MVS/370 versions of the downward incompatible macros. In some cases the new parameters are ignored, in other cases they cause assembly errors. The following macro instructions are the downward incompatible macros described in this book:

- ATTACH
- ESTAE
- EVENTS
- FESTAE
- INTSECT
- SCHEDULE SCOPE=GLOBAL
- SDUMP
- SETLOCK RELEASE,TYPE=REG|ALL
- WTOR

The SPLEVEL macro instruction solves the problems associated with downward incompatible macros. The SPLEVEL macro instruction allows an installation to assemble programs using the MVS/XA macro library and to select either the MVS/370 System Product Version 1 Release 3 or the MVS/XA expansion of the downward incompatible macros.

Before issuing a downward incompatible macro, assembler language users can specify the macro level that they want. They do this by issuing the SPLEVEL macro using the SET=*n* option, with *n*=1 or 2. If *n*=1, the MVS/370 System Product Version 1 Release 3 expansion of the macro code is generated and if *n*=2, the MVS/XA expansion of the macro code is generated. If the user does not specify the value of *n*, the SPLEVEL routine uses the default value of 2. See *SPL: System Modifications* for information concerning the way in which an installation can set this default.

A user can also select the level of the macro at execution time, based on the system that is operating. The example in Figure 1 shows one method of selecting the macro level at execution time. The example uses the WTOR macro instruction, but any downward incompatible macro instruction could be substituted. The code makes use of the fact that the CVTMVSE bit in byte CVTDCB (located at offset 116 or X'74' of the communications vector table (CVT)) is set to 1 when MVS System Product Version 2 is operating. The CVTMVSE field of the CVT is defined in System Product Version 2.

```
*     DETERMINE WHICH SYSTEM IS EXECUTING
          TM        CVTDCB,CVTMVSE
          BO        SP2
*     INVOKE MVS/370 VERSION OF THE MACRO
          SPLEVEL   SET=1
          WTOR      ...
          B         CONTINUE
*     INVOKE MVS/XA VERSION OF THE MACRO
SP2       SPLEVEL   SET=2
          WTOR      ...
*     RESET TO SYSTEM DEFAULT
CONTINUE SPLEVEL   SET
```

Figure 1. Macro Level Selected at Execution Time

Addressing Mode and the Macro Instructions

Callers in either 24-bit or 31-bit addressing mode can invoke most of the macros described in this book. The following is a list of the macro instructions, documented in this book, that require the caller to be executing in 24-bit addressing mode and require that the parameters be located in 24-bit addressable storage:

RACDEF
RACHECK
RACINIT
RACLIST
SPIE
STAE

Note: RACF services are also available through the RACROUTE macro, which can execute in either 24-bit or 31-bit addressing mode.

In general, a program executing in 24-bit addressing mode cannot pass parameters located above 16 megabytes in virtual storage to a system service. There are exceptions to this general rule. For example, a program executing in 24-bit addressing mode can:

- Free storage above 16 megabytes using the FREEMAIN macro instruction
- Allocate storage above 16 megabytes using the GETMAIN macro instruction
- Perform cell pool services for cell pools located in storage above 16 megabytes using the CPOOL macro instruction
- Perform page services for storage locations above 16 megabytes using the PGSER macro instruction

See the descriptions of the individual macro instructions for details.

If a program is executing in 31-bit addressing mode, the addresses specified as parameters for the macro instructions in this book can be located above or below the 16 megabytes line unless otherwise stated. If a parameter passed by a program executing in 31-bit addressing mode must be located below the 16 megabytes line in virtual storage, the restriction is stated in the description of the parameter of the macro instruction.

If you are executing in 31-bit addressing mode, you must use the MVS/XA version of the following macro instructions:

ATTACH
CALLDISP
ESTAE
EVENTS
FESTAE
INTSECT
MODESET
SETRP
SNYCH
WTOR

Cross Memory Restrictions for Macro Instructions

The topic "Cross Memory" in Volume 1 describes the general restrictions pertaining to cross memory and the general functions available to callers in cross memory mode. Unless stated, a macro service is not available in cross memory mode. A brief description of how specific macro instructions can be used in cross memory is given here.

The following macro instructions are available to callers in cross memory mode without restrictions:

ABEND
DSGNL
INTSECT (global intersect)
LOCASCB (locate an ASCB from an ASID)
RISGNL
RPSGNL
PTRACE
SCHEDULE
SETLOCK (for global locks)
SUSPEND
VSMREGN (provides addresses in the current address space)

The following services have special options or restrictions for cross memory mode programs:

ATSET - The issuer of this macro instruction must be executing in primary mode.

ATEXT - The issuer of this macro instruction must be executing in primary mode.

AXFRE - The issuer of this macro instruction must be executing in primary mode.

AXRES - The issuer of this macro instruction must be executing in primary mode.

AXSET - The issuer of this macro instruction must be executing in primary mode.

CALLDISP - This macro instruction is available if the caller uses the BRANCH=YES option.

CALLRTM - This macro instruction has options and restrictions related to cross memory.

CPOOL - This macro instruction is available to all cross memory callers, except for callers in secondary mode, who specify LINKAGE=SYSTEM.

CPUTIMER - This macro instruction can be invoked in primary cross memory mode.

ETCON - The issuer of this macro instruction must be executing in primary mode.

ETCRE - The issuer of this macro instruction must be executing in primary mode.

ETDES - The issuer of this macro instruction must be executing in primary mode.

ETDIS - The issuer of this macro instruction must be executing in primary mode.

GETMAIN/FREEMAIN (private storage) - The GETMAIN/FREEMAIN macro instructions with the BRANCH=YES option can be used in cross memory mode to obtain

private storage if the caller has current addressability to the address space and holds the address space's local lock as a CML lock.

GETMAIN/FREEMAIN (common storage) - The GETMAIN/FREEMAIN macro instruction with the BRANCH=(YES,GLOBAL) option is available in cross memory mode to obtain common storage.

GQSCAN - The issuer of this macro instruction must be executing in primary mode.

LXFRE - The issuer of this macro instruction must be executing in primary mode.

LXRES - The issuer of this macro instruction must be executing in primary mode.

MODESET - The inline form of the MODESET macro instruction can by used by any callers in cross memory mode.

PCLINK - The STACK and UNSTACK options are available to issuers in primary mode. The EXTRACT option is available to a caller with addressability to the same address space as when PCLINK STACK was issued for the stack element from which data is being extracted.

PGFIX/PGFREE - These macro instructions have restrictions related to cross memory. See the description of the individual macro instruction for details.

PGSER - The ANYWHER, FIX, FREE, LOAD, OUT, and RELEASE options of this macro are available to an enabled caller in supervisor state, key zero, who specifies branch entry. To use the LOAD and the ANYWHER options, the issuer of PGSER must not be running in secondary mode.

RESUME - To issue RESUME, the requestor must have current addressability to the address space of the task being resumed. That is, the address space must be the current address space.

SDUMP - MVS/XA dumping services format additional data required by cross memory. The SDUMP macro instruction with the BRANCH=YES option is supported in cross memory mode, and other options dump address spaces related to the failing address space.

SETFRR - The SETFRR macro instruction can set up a recovery environment in cross memory mode and provides predictable entry and re-try environments in case of error.

SETLOCK (CML lock) - Programs can call the MVS/XA lock manager using the SETLOCK macro instruction. The program can request the local lock of another address space (the CML lock) in order to serialize resources in the other address space. The requestor must have an active addressing bind to the address space whose local lock he is requesting.

SETRP - This macro instruction supports the freeing of the CML lock when a functional recovery routine requests that termination processing continue, and it also has an improved mechanism to get from SRB recovery to related task recovery. SETRP also supports a cross memory mode re-try environment.

SLIP - The operator can set SLIP traps to intercept an event in cross memory mode.

SRBSTAT - Callers must have the authority to issue a SSAR to the home address space. The save area must be addressable from the home address space. Control returns from the SRBSTAT macro instruction in primary mode.

SSAFF - TCB subsystem affinity - This macro instruction, described in *SPL: System Modifications*, has restrictions associated with cross memory.

WAIT/POST - The WAIT and cross address space POST branch entry services provide restricted support.

VSMLIST and VSMLOC - Callers who specify LINKAGE=SYSTEM cannot be in secondary mode. All address returned by these macro instructions are associated with the current address space.

See the topic "Summary of MVS/XA Facilities Available in Cross Memory Mode" in Volume 1 for other functions that are available to callers in cross memory mode.

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 a 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 you cannot use registers in the list form.

Use the execute form of a 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. If you do not generate the control program parameter list using the list form of the macro, you must provide the list yourself, initialize it, then update it directly or by explicitly specifying keywords on the execute form.

Some macros also provide a modify form. Use the modify form of a macro instruction to modify a parameter list created with the list form of the macro instruction.

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 *Assembler H Version 2 Application Programming: Language Reference*.

Coding the Macro Instructions

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

Figure 2 shows a sample macro instruction, TEST, and summarizes all the coding information that is available for it. The table is divided into three columns, A, B, and C.

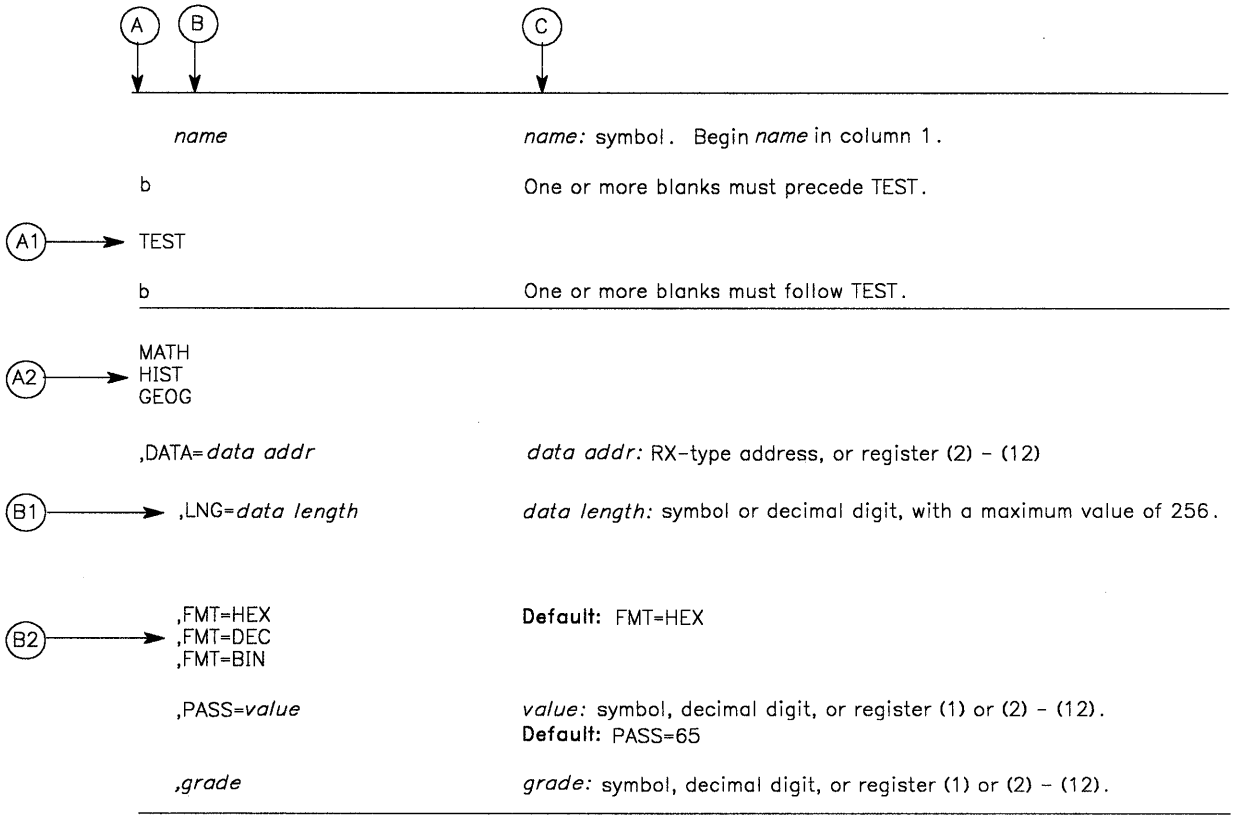


Figure 2. Sample Macro Instruction

- 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 you must code it. If two or more lines appear together, A2 , you must code the parameter appearing on one and only one of the lines.

- 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, code one and only one of the lines.

- The third column, C , provides additional information for coding the macro instruction. When substitution of a variable is required, the following classifications are used:

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. You must set the unused high-order bits to zero. You can designate the register 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 can be written in an A-type address constant.

default: a value that is used in default of a specified value; that is, the value 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, MATH, HIST, or FMT=HEX), code the parameter exactly as shown.
- If the selected parameter is written in italics (for example, *grade*), 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, DATA=*data addr*), 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 the third column before proceeding to the next parameter. The third column often contains coding restrictions for 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 3 shows an example of each method.

```
1         10        16                                  44                                  72
↓         ↓         ↓                                   ↓                                   ↓
NAME1     OP1       OPERAND1,OPERAND2,OPERAND3,OPERAND4,OPERAND5,OPERAND6,OPX
                    ERAND7                              THIS IS ONE WAY
NAME2     OP2       OPERAND1,OPERAND2,                  THIS IS ANOTHER WAY             X
                    OPERAND3,OPERAND4,                                                  X
                    OPERAND5,OPERAND6,OPERAND7
```

Figure 3. Continuation Coding

ATSET - Set Authorization Table

The ATSET macro instruction sets both the PT and SSAR authority in the home address space's authorization table entry that corresponds to the specified authorization index (AX) value. This action sets up authority for address spaces with the specified AX to issue a PT instruction (PT=YES) or SSAR instruction (SSAR=YES) into the home address space.

The caller must be either in supervisor state or PKM 0-7, executing in primary mode enabled and unlocked.

Before entry to this macro, register 13 must point to a standard register save area addressable in primary mode. Register 2, which is modified by the macro after the registers are saved, should not be used as the base register.

After completion, the registers contain the following information:

- Registers 0 and 1 are unpredictable.
- Registers 2 - 14 are preserved.
- Register 15 contains the return code.

The ATSET macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ATSET.
ATSET	
b	One or more blanks must follow ATSET.
AX=*AX value*	*AX value:* RX-type address or general register (0) - (12).
,PT=NO ,PT=YES	**Default:** PT=NO
,SSAR=NO ,SSAR=YES	**Default:** SSAR=NO
,RELATED=*value*	*value:* any valid macro instruction keyword specification.

The parameters are explained as follows:

AX = *AX value*
specifies the AX value for which the PT and SSAR authority are to be set. If the RX-type address is used, it points to the address of a half word, addressable in primary mode, that contains the AX value. If the register form is used, the AX value must be in bits 16-31; bits 0-15 are ignored.

,PT = NO
,PT = YES
specifies whether (YES) or not (NO) a program transfer (PT) into the home address space by routines executing with the specified AX is to be allowed.

,SSAR = NO
,SSAR = YES
specifies whether (YES) or not (NO) routines executing with the specified AX are to be allowed to establish secondary addressability to the home address space.

,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 can be any valid coding values.

Note: Both the PT and SSAR authority are set every time you invoke the ATSET macro instruction. If you do not specify PT, for example, the PT authority is set off. If you want the PT authority to remain on, you must specify PT = YES.

When control returns, register contains the following return code:

Hexadecimal Code	Meaning
0	The selected authorization table entry has been set

ATTACH - Create a New Task

This macro can be assembled compatibly between MVS/XA and MVS/370 through the use of the SPLEVEL macro instruction. Default processing will result in an expansion of the macro that operates only with MVS/XA. See the topic "Selecting the Macro Level" for additional information. If you are executing in 31-bit addressing mode, you must use the MVS/XA version of this macro instruction.

The ATTACH macro instruction causes the control program to create a new task and indicates the entry point in the program to be given control when the new task becomes active. The entry point name that is specified must be a member name or an alias in a directory of a partitioned data set, or must have been specified in an IDENTIFY macro instruction. If the specified entry point cannot be located, the new subtask is abnormally terminated.

On entry to the attached routine, the high order bit, bit 0, of register 14 is set to indicate the addressing mode of the issuer of the ATTACH macro. If bit 0 is 0, the issuer is executing in 24-bit addressing mode; if bit 0 is 1, the issuer is executing in 31-bit addressing mode.

The address of the task control block for the new task is returned in register 1. The new task is a subtask of the originating task; the originating task is the task that was active when the ATTACH macro instruction was issued. The limit and dispatching priorities of the new task are the same as those of the originating task unless modified in the ATTACH macro instruction.

The load module containing the program to be given control is brought into virtual storage if a usable copy is not available in virtual storage. The issuing program can provide an event control block, in which termination of the new task is posted, an exit routine to be given control when the new task is terminated, and a parameter list whose address is passed in register 1 to the new task. If you code neither the ECB nor ETXR parameter, the subtask is automatically removed from the system upon completion of its execution. If you specify the ECB parameter in the ATTACH macro instruction, the ECB must be in storage so that you can wait on it (using the WAIT macro instruction) and the control program can post it on behalf of the terminating task. You can also use the ATTACH macro instruction to specify that ownership of virtual subpools is to be assigned to the new task, or that the subpools are to be shared by the originating task and the new task.

Except for DCB and JSCB, all input parameters to the ATTACH macro instruction can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

The description of the ATTACH macro instruction follows. The ATTACH macro instruction is also described in *Supervisor Services and Macro Instructions* with the exception of the JSTCB, SM, SVAREA, KEY, DISP, JSCB, TID, NSHSPV, NSHSPL, and RSAPF parameters. These parameters are restricted in use to supervisor state or PSW key 0-7 programs and, therefore, are only described here.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ATTACH.
ATTACH	
b	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).
,RSAPF=NO ,RSAPF=YES	**Default:** RSAPF=NO
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained below:

EP = *entry name*
EPLOC = *entry name addr*
DE = *list entry addr*

specifies the entry name, the address of the entry name, or the address of the name field of a 60-byte list entry for the entry name that was constructed using the BLDL macro instruction. If EPLOC is coded, the name must be padded to eight bytes, if necessary.

Notes:

1. *ATTACH processing can attach a load module in 24-bit or 31-bit addressing mode physically resident above or below 16 megabytes virtual. The AMODE and RMODE, which are load module attributes located in the directory entry for the load module, provide this information. The RMODE indicates where the module is to be placed; the AMODE indicates the addressing mode of the module. If the AMODE of the entry point being attached is ANY, it will be attached with the same addressing mode as the caller.*

2. *The task structure must not be changed via an ATTACH or DETACH between the issuance of the BLDL and the issuance of the ATTACH for the module, or an abend 106 with a return code of 15 might result.*

,DCB = *dcb addr*

specifies the address of the data control block for the partitioned data set containing the entry name described above.

Note: The DCB must be opened before the ATTACH macro instruction is executed and must reside in storage below 16 megabytes.

,LPMOD = *limit prior nmbr*

specifies the number (255 or less) to be subtracted from the current limit priority of the originating task. The result is the limit priority of the new task. If this parameter is omitted, the current limit priority of the originating task is assigned as the limit priority of the new task.

,DPMOD = *disp prior nmbr*

specifies the signed number (255 or less) to be algebraically added to the current dispatching priority of the originating task. The result is assigned as the dispatching priority of the new task, unless it is greater than the limit priority of the new task. If the result is greater, the limit priority is assigned as the dispatching priority.

If a register is designated, a negative number must be in two's complement form in the register. If this parameter is omitted, the dispatching priority assigned is the smaller of either the new task's limit priority or the originating task's dispatching priority.

,PARAM = (*addr*)
,PARAM = (*addr*), **VL** = **1**

specifies the address(es) to be passed to the attached program. Each address is expanded inline to a fullword on a fullword boundary, in the order designated. Register 1 contains the address of the first word when the program is given control.

VL=1 should be designated only if the called program can be passed a variable number of parameters. VL=1 causes the high-order bit of the last address to be set to 1; the bit can be checked to find the end of the list.

,ECB=*ecb addr*
specifies the address of an event control block for the new task to be used by the control program to indicate the termination of the new task. The ECB must be in storage so that the issuer of the attach can wait on it (using the WAIT macro instruction) and the control program can post it on behalf of the terminating task. The return code (if the task is terminated normally) or the completion code (if the task is terminated abnormally) is also placed in the event control block. If this parameter is coded, a DETACH macro instruction must be issued to remove the subtask from the system after the subtask has been terminated.

,ETXR=*exit rtn addr*
specifies the address of the end-of-task exit routine to be given control after the new task is normally or abnormally terminated. The exit routine is given control when the originating task becomes active after the subtask is terminated, and must be in virtual storage when required. If this parameter is coded, a DETACH macro instruction must be issued to remove the subtask from the system after the subtask has been terminated.

The exit routine receives control in the addressing mode of the caller of the ATTACH macro instruction. ATTACH processing issues an ABEND with completion code X'72A' if a caller attempts to create two subtasks with the same exit routine in different addressing modes.

The contents of the registers when the exit routine is given control are as follows:

Register	Contents
0	Control Program Information
1	Address of the task control block for the task that was terminated
2-12	Unpredictable
13	Address of a save area provided by the control program
14	Return address (to the control program)
15	Address of the exit routine

The exit routine is responsible for saving and restoring the registers.

,GSPV=*subpool nmbr*
,GSPL=*subpool list addr*
specifies a virtual storage subpool number less than 128 or the address of a list of virtual storage subpool numbers each of which less than 128. Except for subpool 0, ownership of each of the specified subpools is assigned to the new task. Although you can specify subpool zero, it cannot be transferred. When a task transfers ownership of a subpool, it can no longer GETMAIN or FREEMAIN the associated virtual storage areas.

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

,SHSPV=*subpool nmbr*
,SHSPL=*subpool list addr*
specifies a virtual storage subpool number less than 128 or the address of a list of virtual storage subpool numbers each less than 128. Programs of both originating task and the new task can use the associated virtual storage areas.

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

,SZERO=YES
,SZERO=NO
specifies whether subpool 0 is to be shared with the subtask. YES specifies that subpool 0 is to be shared; NO specifies that subpool 0 is not to be shared.

,TASKLIB=*dcb addr*
specifies that a task library DCB address has been supplied and is to be stored in TCBJLB. Otherwise, TCBJLB is propagated from the originating task. If the DCB address specifies LINKLIB, no other library is searched because searching LINKLIB indicates the end of the search.

Note: The DCB must be opened before the ATTACH macro instruction is executed and must reside in storage below 16 megabytes.

,STAI=(*exit addr***)**
,STAI=(*exit addr,parm addr***)**
,ESTAI=(*exit addr***)**
,ESTAI=(*exit addr,parm addr***)**
specifies whether a STAI or ESTAI SCB is to be created; any STAI/ESTAI SCBs queued to the originating task are propagated to the new task.

The *exit addr* specifies the address of the STAI or ESTAI exit routine which is to receive control if the subtask abnormally terminates; the exit routine must be in virtual storage at the time of abnormal termination. The *parm addr* is the address of a parameter list that can be used by the STAI or ESTAI exit routine.

ATTACH processing passes control to the ESTAI exit routine in the addressing mode of the caller of the ATTACH service routine. Therefore, the ESTAI exit routine can execute in either 24-bit or 31-bit addressing mode. A STAI exit routine can execute only in 24-bit addressing mode. If a caller in 31-bit addressing mode specifies the STAI parameter on the ATTACH macro instruction, the caller is abended with an X'52A' completion code.

,PURGE=QUIESCE
,PURGE=NONE
,PURGE=HALT
specifies what action is to be taken with regard to I/O operations when the subtask is abnormally terminated. No action may be specified (NONE), a halting of I/O operations may be requested (HALT), or a quiescing of I/O operations may be indicated (QUIESCE).

,ASYNCH=NO
,ASYNCH=YES
specifies whether asynchronous exits are to be allowed when a subtask abnormal termination occurs.

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 ESTAI 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 ESTAI 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.

,TERM=NO
,TERM=YES
specifies whether the exit routine associated with the ESTAI request is also to be scheduled in the following situations:

- CANCEL

- Forced LOGOFF

- Job step timer expirations

- Wait time limit for job step exceeded

- ABEND condition because incomplete task detached when STAE option not specified on DETACH

- Attaching task abnormally terminates

,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.

Note: The JSTCB=YES option causes a new job pack area to be established for the attached task. Any modules within the job pack area of the task issuing the ATTACH are therefore not implicitly known to the newly attached 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 obtains 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.

Note: The JSCB parameter must specify a storage location below 16 megabytes.

,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.

,RSAPF = YES
specifies that the attached subtask may come from an unauthorized library. If, however. it comes from an APF-authorized library and is link-edited with the APF-authorized attribute, the step begins execution with APF authorization.

RSAPF = YES applies when all of the following conditions are met:

- The caller is running in supervisor state, system key (0-7), or both.
- The caller is running non-APF authorized.
- The task is attached in the problem program state and with a non-system key.

,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.

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.

Notes:

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

2. *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.*

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 save area 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 save area 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,    X
  JSCB=(5),DISP=NO,TASKLIB=(8)
```

Example 3

Operation: Cause the program named in the list to be attached. Establish RTN as an end of task exit routine.

```
ATTACH  DE=LISTNAME,ETXR=RTN
```

Example 4

Operation: Cause PROGRAM1 to be attached, share subpool 5, supply WORD1 so the originating task can know when the subtask is complete, and establish EXIT1 as an ESTAI exit.

```
ATTACH   EP=PROGRAM1,SHSPV=5,ECB=WORD1,ESTAI=(EXIT1)
```

ATTACH (List Form)

Two parameter lists are used in an ATTACH macro instruction: a control program parameter list and a problem program parameter list. You can construct only the control program parameter list in the list form of ATTACH. Address parameters to be passed in a parameter list to the problem program can be provided using the list form of the CALL macro instruction. This parameter list can be referred to in the execute form of ATTACH.

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

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

EP=*entry name* EPLOC=*entry name addr* DE=*list entry addr*	*entry name:* symbol. *entry name addr:* A-type address. *list entry addr:* A-type address.
,DCB=*dcb addr*	*dcb addr:* A-type address.
,LPMOD=*limit prior nmbr*	*limit prior nmbr:* symbol or decimal digit.
,DPMOD=*disp prior nmbr*	*disp prior nmbr:* symbol or decimal digit.
,ECB=*ecb addr*	*ecb addr:* A-type address.
,ETXR=*exit rtn addr*	*exit rtn addr:* A-type address.
,GSPV=*subpool nmbr* ,GSPL=*subpool list addr*	*subpool nmbr:* symbol or decimal digit. *subpool list addr:* A-type address.
,SHSPV=*subpool nmbr* ,SHSPL=*subpool list addr*	*subpool nmbr:* symbol or decimal digit. *subpool list addr:* A-type address.
,SZERO=YES ,SZERO=NO	**Default:** SZERO=YES
,TASKLIB=*dcb addr*	*dcb addr:* A-type address.
,STAI=*(exit addr)* ,STAI=*(exit addr,parm addr)* ,ESTAI=*(exit addr)* ,ESTAI=*(exit addr,parm addr)*	*exit addr:* A-type address. *parm addr:* A-type address.
,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 specified 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.
,TID=*task id*	*task id:* decimal digits 0-255. **Default:** TID=0
,NSHSPV=*subpool nmbr* ,NSHSPL=*subpool list addr*	*subpool nmbr:* symbol, decimal digit. *subpool list addr:* A-type address.
,RSAPF=NO ,RSAPF=YES	**Default:** RSAPF=NO
,RELATED=*value*	*value:* any valid macro keyword specification.
,SF=L	

The parameters are explained under the standard form of the ATTACH macro instruction, with the following exception:

,SF=L
specifies the list form of the ATTACH macro instruction.

Note: If RSAPF parameter is not specified on the list form of the ATTACH macro instruction, the default is RSAPF=NO. If RSAPF=YES is specified on the list form or on a previous execute form using the same SF= list, RSAPF=NO is ignored for any subsequent execute forms of the ATTACH macro instruction.

Once RSAPF is specified, it is in effect for all users of that list.

ATTACH (Execute Form)

Two parameter lists are used in ATTACH: a control program parameter list and an optional problem program parameter list. Either or both of these parameter lists can be remote and can be referred to and modified by the execute form of ATTACH. If only the problem program parameter list is remote, parameters that require use of the control program parameter list cause that list to be constructed inline as part of the macro expansion.

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

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

EP=*entry name*	*entry name:* symbol.
EPLOC=*entry name addr*	*entry name addr:* RX-type address, or register (2) - (12).
DE=*list entry addr*	*list entry addr:* RX-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr:* RX-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)*	*addr:* RX-type address, or register (2) - (12).
,PARAM=*(addr),VL=1*	**Note:** *addr* is one or more addresses, separated by commas. For example, PARAM=*(addr,addr,addr)*
,ECB=*ecb addr*	*ecb addr:* RX-type address, or register (2) - (12).
,ETXR=*exit rtn addr*	*exit rtn addr:* RX-type address, or register (2) - (12).
,GSPV=*subpool nmbr*	*subpool nmbr:* symbol, decimal digit, or register (2) - (12).
,GSPL=*subpool list addr*	*subpool list addr:* RX-type address, or register (2) - (12).
,SHSPV=*subpool nmbr*	*subpool nmbr:* symbol, decimal digit, or register (2) - (12).
,SHSPL=*subpool list addr*	*subpool list addr:* RX-type address, or register (2) - (12).
,SZERO=YES	
,SZERO=NO	
,TASKLIB=*dcb addr*	*dcb addr:* RX-type address, or register (2) - (12).
,STAI=*(exit addr)*	*exit addr:* RX-type address, or register (2) - (12).
,STAI=*(exit addr,parm addr)*	*parm addr:* RX-type address, or register (2) - (12).
,ESTAI=*(exit addr)*	
,ESTAI=*(exit addr,parm addr)*	
,PURGE=QUIESCE	**Note:** PURGE may be specified only if STAI or ESTAI is specified.
,PURGE=NONE	
,PURGE=HALT	
,ASYNCH=NO	**Note:** ASYNCH may be specified only if STAI or ESTAI is specified.
,ASYNCH=YES	
,TERM=NO	**Note:** TERM may be specified only if ESTAI is specified.
,TERM=YES	
,JSTCB=NO	**Default:** JSTCB=NO
,JSTCB=YES	
,SM=PROB	**Default:** SM=PROB
,SM=SUPV	
,SVAREA=YES	
,SVAREA=NO	**Default:** SVAREA=YES
,KEY=PROP	**Default:** KEY=PROP
,KEY=ZERO	
,DISP=YES	**Default:** DISP=YES
,DISP=NO	
,JSCB=*jscb addr*	*jscb addr:* RX-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*	*subpool nmbr:* symbol, decimal digit, or register (2) - (12).
,NSHSPL=*subpool list addr*	*subpool list addr:* RX-type address, or register (2) - (12).
,RSAPF=NO	**Default:** RSAPF=NO
,RSAPF=YES	
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=*(E,prob addr)*	*prob addr:* RX-type address, or register (1) or (2) - (12).
,SF=*(E,ctrl addr)*	*ctrl addr:* RX-type address, or register (2) - (12) or (15).
,MF=*(E,prob addr),SF=(E,ctrl addr)*	

The parameters are explained under the standard form of the ATTACH macro instruction, with the following exceptions:

,MF=(E, *prob addr***)**
,SF=(E, *ctrl addr***)**
,MF=(E, *prob addr***) ,SF=(E,** *ctrl addr***)**
specifies the execute form of the ATTACH macro instruction using a remote problem program parameter list or a remote control program parameter list or both. If a parameter list is not provided, any problem program or control program parameters are provided in parameter lists expanded inline.

Notes:

1. *If STAI is specified on the execute form, the following fields are overlaid in the control program parameter list: exit addr, parm addr, PURGE, and ASYNCH. If parm addr is not specified, zero is used; if PURGE or ASYNCH are not specified, defaults are used.*

2. *If ESTAI is specified on the execute form, the following fields are overlaid; exit addr, parm addr, PURGE, ASYNCH, and TERM. If parm addr is not specified, zero is used; if PURGE, ASYNCH, or TERM are not specified, defaults are used.*

3. *If the STAI or ESTAI is to be specified, it must be completely specified on either the list or execute form, but not on both forms.*

4. *If SZERO is not specified on the list or execute form, the default is SZERO=YES. If SZERO=NO is specified on either the list form or a previous execute form using the same SF=list, then SZERO=YES is ignored for any following execute forms of the macro. Once SZERO=NO is specified, it is in effect for all users of that list.*

5. *If RSAPF=YES is specified on the list form of the ATTACH macro instruction or on a previous execute form of the ATTACH macro instruction using the same SF= list, RSAPF=NO is ignored for any subsequent execute forms of the ATTACH macro instruction.*

AXEXT - Extract Authorization Index

The AXEXT macro instruction returns to the caller the authorization index (AX) value of the specified address space.

The caller must be in supervisor state or PKM 0-7 executing in primary mode enabled and unlocked. Register 13 must point to a standard register save area that must be addressable in primary mode.

Registers 2-14 are preserved. Register 2, which is modified by the macro after the registers are saved, should not be used as the base register. Register 15 contains the return code. The extracted AX is placed in bits 16-31 of register 0 and bits 0-15 are set to zero. The contents of register 1 are unpredictable.

The AXEXT macro instruction is written as follows:

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

ASID=*asid value*	*asid value:* RX-type address or register (0) - (12). **Default:** current PASID.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

ASID=*asid value*
specifies the ASID of the address space whose AX is to be extracted. If the RX-type address is used, it points to a halfword containing the ASID. If the register form is used, the register must contain the ASID in bits 16-31 with bits 0-15 set to zero. If ASID is not specified, the current PASID is assumed.

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

When control returns, register 15 contains the following return code:

Hexadecimal Code	Meaning
0	The AX value of the specified address space was successfully obtained.

AXFRE - Free Authorization Index

The AXFRE macro instruction returns one or more authorization index (AX) values to the system. The caller must ensure that the AXs to be returned are no longer being used by any address space or else the caller is abnormally terminated. On completion of the AXFRE macro instruction, all authorization of the freed AX values in authorization tables for the entire system will be purged. The caller must be dispatched in the address space that owns the AX.

The caller must be in supervisor state or PSW mask 0-7, executing in primary mode enabled and unlocked.

Register 13 must point to a standard register save area that must be addressable in primary mode. The list of AX values passed to the AXFRE macro instruction must also be addressable in primary mode at the time the macro instruction is issued.

Registers 2-12 are preserved. Register 2, which is modified by the macro after the registers are saved, should not be used as the base register. Register 15 contains the return code. The contents of registers 0 and 1 are unpredictable.

The AXFRE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One ore more blanks must precede AXFRE.
AXFRE	
b	One or more blanks must follow AXFRE.

AXLIST=*list addr*	*list addr:* RX-type address or register (0) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

AXLIST=*list addr*
specifies the address of a variable length list of halfword entries that contain the AX values to be freed. The first halfword must contain the number of values in the list.

,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 can be any valid coding values.

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

Hexadecimal Code	Meaning
0	The specified authorization index or indexes were successfully freed.
4	The specified authorization index or indexes were not successfully freed. One or more of the indexes could be unavailable for use.

AXRES - Reserve Authorization Index

The AXRES macro instruction reserves one or more authorization index (AX) values for the caller's use. The AX values are then owned by the current home address space.

The caller must be in supervisor state or PKM 0-7, executing in primary mode enabled and unlocked. The parameter list passed to the AXRES macro instruction must be addressable in primary mode at the time the macro expansion is executed. Register 13 must point to a standard register save area that must be addressable in primary mode.

Registers 2-14 are preserved. Register 2, which is modified by the macro after the registers are saved, should not be used as the base register. Register 15 contains the return code. The contents of registers 0 and 1 are unpredictable.

The AXRES macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede AXRES.
AXRES	
b	One or more blanks must follow AXRES.
AXLIST=*list addr*	*list addr:* RX-type address or register (0) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

AXLIST=*list addr*
specifies the address of a variable length list, addressable in primary mode, of halfword entries in which the requested AX values are to be returned. The first halfword must contain the number of values to be returned. Enough halfwords must follow the first entry to contain the requested number of values. If the requested number of AX values is not available, the caller is abnormally terminated.

,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 can be any valid coding values.

When control returns, register 15 contains the following return code.

Hexadecimal Code	Meaning
0	The AX value or values were successfully reserved.

AXSET - Set Authorization Index

The AXSET macro instruction sets the authorization index (AX) of the home address space to the value specified by the caller. The AX must have been previously reserved and the address space whose AX is being changed cannot own connected space switch entry tables. All routines that subsequently execute with a PASID of the address space whose AX was changed execute with the new AX.

The caller must be in supervisor state or PKM 0-7 executing in primary mode enabled and unlocked. Register 13 must point to a standard register save area that must be addressable in primary mode.

Registers 2-14 are preserved. Register two, which is modified by the macro after the registers are saved, should not be used as the base register. Register 0 contains the original AX value in bits 16-31 with bits 0-15 set to zero. Register 15 contains the return code. The contents of register 1 are unpredictable.

The AXSET macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede AXSET.
AXSET	
b	One or more blanks must follow AXSET.
AX=*AX value*	*AX value:* RX-type address or register (0) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

AX=AX *value*
specifies the new AX value. The RX-type address specifies a halfword containing the new AX. If the register form is used, the register must contain the new AX in bits 16-31 and bits 0-15 must be zero.

,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 can be any valid coding values.

When control returns, register 15 contains the following return code:

Hexadecimal Code	Meaning
0	The AX of the home address space was set to the value specified by the caller.

BLSABDPL - Map the Exit Parameter List BLSABDPL

The BLSABDPL macro instruction maps the exit parameter list (BLSABDPL), which is a data area that enables IPCS, PRDMP, SNAP, and user-written exit routines to tailor dumps.

Using this macro, you can map the following areas within the BLSABDPL exit parameter list:

- The processor status record
- The storage access parameter list
- The select ASID parameter list
- The control block and format model processor parameter list
- The ECT parameter list

By accessing any one of these parameter lists, the exit routine can then use the data in the parameter list to invoke the corresponding exit service routine. For information about using the exit service routines, see *MVS/XA Interactive Problem Control System User's Guide and Reference.*

The BLSABDPL macro instruction is written as follows:

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

AMDCPST=YES AMDCPST=NO	**Default:** AMDCPST=NO
,AMDEXIT=YES ,AMDEXIT=NO	**Default:** AMDEXIT=YES
,AMDOSEL=YES ,AMDOSEL=NO	**Default:** AMDOSEL=YES
,AMDPACC=YES ,AMDPACC=NO	**Default:** AMDPACC=YES
,AMDPECT=YES ,AMDPECT=NO	**Default:** AMDPECT=YES
,AMDPFMT=YES ,AMDPFMT=NO	**Default:** AMDPFMT=YES
,AMDPSEL=YES ,AMDPSEL=NO	**Default:** AMDPSEL=YES
,DSECT=YES ,DSECT=NO	**Default:** DSECT=YES

The parameters are explained as follows:

AMDCPST=YES
AMDCPST=NO
specifies whether the format of the CPU status data available through the IPCS and PRDMP storage access services is to be mapped (YES) or suppressed (NO).

If this parameter is not specified, the default is NO.

Because the system uses DSECT AMDCPMAP to map the format of CPU status data (when AMDCPST = YES), the system ignores the DSECT = NO option if it is specified.

AMDEXIT=YES
AMDEXIT=NO
specifies whether the common exit parameter list (BLSABDPL) is to be mapped (YES) or suppressed (NO).

If this parameter is not specified, the default is YES.

The common exit parameter list contains two parts: ABDPL and

ADPLEXTN. DSECT=YES causes DSECT statements to be generated for both. DSECT=NO suppresses the DSECT statements and causes ABDPL and ADPLEXTN to be defined as the labels associated with the first bytes described in the ABDPL and ADPLEXTN exit parameter lists, respectively.

AMDOSEL=YES
AMDOSEL=NO
specifies whether the select ASID service output data available under IPCS and PRDMP is to be mapped (YES) or suppressed (NO).

If this parameter is not specified, the default is YES.

If the DSECT=NO option is specified, it is ignored. The select ASID parameter list is always mapped using DSECT ADPLPSEL.

Because the system uses DSECT ADPLPSEL to map the select ASID parameter list (when AMDOSEL = YES), the system ignores the DSECT = NO option if it is specified.

AMDPACC=YES
AMDPACC=NO
specifies whether the storage access service parameter list is to be mapped (YES) or suppressed (NO).

If this parameter is not specified, the default is YES.

The storage access service parameter list is described as ADPLPACC. DSECT=YES causes DSECT statements to be generated for ADPLPACC. DSECT=NO suppresses the DSECT statements and causes ADPLPACC to be defined as the label associated with the first byte described in the storage access service parameter list.

AMDPECT=YES
AMDPECT=NO
specifies whether the ECT service parameter list is to be mapped (YES) or suppressed (NO).

If this parameter is not specified, the default is YES.

The ECT service parameter list is described as ADPLPECT. DSECT=YES causes DSECT statements to be generated for ADPLPECT. DSECT=NO suppresses the DSECT statements and causes ADPLPECT to be defined as the label associated with the first byte described in the ECT service parameter list.

AMDPFMT=YES
AMDPFMT=NO
specifies whether the parameter list used by both the control block formatter and the format model processor services is to be mapped (YES) or suppressed (NO).

If this parameter is not specified, the default is YES.

The parameter list used by both the control block formatter and the format model processor services is described as ADPLPFMT. DSECT=YES causes DSECT statements to be generated for ADPLPFMT. DSECT=NO suppresses the DSECT statements and causes ADPLPFMT to be defined as the label associated with the first byte described in the parameter list.

AMDPSEL=YES
AMDPSEL=NO
specifies whether the select ASID service parameter list is to be mapped (YES) or suppressed (NO).

If this parameter is not specified, the default is YES.

The ASID service parameter list is described as ADPLPSEL. DSECT=YES causes DSECT statements to be generated for ADPLPSEL. DSECT=NO suppresses the DSECT statements and causes ADPLPSEL to be defined as the label associated with the first byte described in the ASID service parameter list.

DSECT=YES
DSECT=NO
specifies whether parameter lists mapped by BLSABDPL are to be mapped as DSECTs (YES) or not (NO).

If this parameter is not specified, the default is YES.

NOTE: Output data from services can also be mapped by BLSABDPL. Output data are always mapped as DSECTs. These DSECTs cannot be suppressed by DSECT=NO. To determine whether DSECT=NO can suppress a specific DSECT, see the above parameters.

Example

Operation: Code the macro instructions to invoke the select ASID service routine (that generates a list of selected address spaces within a dump) by reserving space for an initialized select ASID service parameter list and defining the mapping of the ABDPL for the user-written exit routine.

```
BLSADPL DSECT=NO,AMDEXIT=NO,AMDOSEL=NO,AMDPACC=NO,
     AMDPFMT=NO,AMDPECT=NO,AMDPSEL=YES
BLSADPL AMDPACC=NO,AMDPFMT=NO,AMDPECT=NO,AMDPSEL=NO
```

BLSQMDEF - Define a Control Block Format

The BLSQMDEF macro instruction is used to start and end the formatting model of a control block from a dump. A control block model must begin with the BLSQMDEF macro instruction, specifying the appropriate parameters. The end of the model is indicated by a BLSQMDEF macro instruction with only the END keyword specified.

The BLSQMDEF and BLSQMFLD macro instructions work together to create a dump formatting model. A control block model has the following structure:

- One BLSQMDEF macro instruction to begin the model definition.
- At least one BLSQMFLD macro instruction to define the attributes of a desired control block field.
- One BLSQMDEF macro instruction to end the model definition.

The order of the BLSQMFLD statements in the formatting model determines the order the fields printed in the dump. No object code producing assembler statements other than the BLSQMFLD macro instruction should be placed between the BLSQMDEF macro instructions that delimit the start and end of the model definition. The BLSQSHDR macro instruction, which associates text strings with dumped data fields, can be used to clarify the dump for the user.

Through the implementation of BLSQMDEF, BLSQMFLD, and BLSQSHDR users of IPCS, PRDMP, and SNAP can control their dump output within user-written formatting routines. For additional information, refer to *MVS/XA Interactive Problem Control System User's Guide.*

The BLSQMDEF macro instruction is written as follows:

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

END current ,CBLEN = *value*	**Note:** END is required if this BLSQMDEF macro is terminating the format model definition. This is the exclusive use of the END parameter; when END is specified, no other options are allowed.
,BASELBL = *label*	*label:* symbol. *value:* decimal constant, hexadecimal constant, or an absolute value. **Note:** CBLEN is required except when the END parameter is specified.
,MAINTLV = *name*	*name:* 1 to 8 byte character string.
,ACRONYM = *name*	*name:* 1 to 8 byte character string **Note:** If ACRONYM is specified, the ACROLBL or ACROFF parameters should also be specified. If neither are, a default offset of zero is assumed.
,ACROLEN = *value*	*value:* decimal constant, hexadecimal constant, or absolute expression of a number from 1 to 8, inclusive.
,ACROLBL = *label*	*label:* symbol. **Note:** Use ACROLBL only if BASELBL is specified.
,ACROFF = *value*	*value:* decimal constant, hexadecimal constant, or absolute value. **Note:** 1. Use ACROFF if acronym is not at offset zero and BASELBL is not specified 2. The ACROFF value is used when both ACROFF and ACROLBL are specified.
,PREFIX = *value*	*value:* integer constant 0 - 8 inclusive. **Default:** PREFIX = 3
,OFFSETS = PRINT ,OFFSETS = NOPRINT	**Default:** OFFSETS = PRINT
,STRTCOL = *value*	*value:* decimal constant, hexadecimal constant, or an absolute expression. **Default:** STRTCOL = 0
,LBLSPC = *value*	*value:* decimal constant, hexadecimal constant, or an absolute expression. **Default:** LBLSPC = 0
,HEADER = *name*	*name:* one to eight byte character string. **Note:** 1. If HEADER is not specified, ACRONYM value is used. 2. If neither HEADER nor ACRONYM is specified, the control block will not contain a heading.

The parameters are explained as followed:

END

specifies the termination of the control block model. This parameter is required ONLY when the BLSQMDEF macro instruction is used to end the control block format. All other parameters are ignored if this parameter is specified.

BASELBL = *label*

specifies the label of an assembler statement, which is to be used to calculate field offsets. If specified, all field offsets calculated by the BLSQMFLD macro instruction will be relative to this label. If not specified, all field offsets must be explicitly specified on the BLSQMFLD macro instruction via the ACROFF parameter.

CBLEN = *value*

specifies the total length of the control block. Value may be a decimal constant, hexadecimal constant, or an absolute expression. This parameter is required **except** when the END parameter is specified.

MAINTLV = *name*

specifies the maintenance level of the control block. The maintenance level name may be a 1 to 8 byte character string that contains no blanks.

ACRONYM = *name*

specifies the contents of the control block acronym field. Name may be a one to eight byte character string that contains no blanks. If this field is specified, the ACROLBL or ACROFF parameter should also be specified in order to define the offset of the acronym field within the control block. If neither the ACROLBL nor the ACROFF parameter is specified, an offset of zero is assumed.

ACROLEN = *value*

specifies the length of the acronym name specified by the ACRONYM parameter in the event that the acronym name requires blanks. If omitted, the length used is the actual length of the name specified in the ACRONYM parameter (without any blanks). Value may be a decimal constant, hexadecimal constant, or absolute expression of a number from zero to eight, inclusive.

ACROLBL = *label*

specifies the label on the assembler statement that defines the acronym field. The label specified here is used with the label provided by BASELBL to calculate the acronym field offset. Use this parameter only if BASELBL is specified. The ACROLBL parameter is ignored if ACROFF is specified.

ACROFF = *value*

specifies the offset of the field containing the control block acronym within the control block. Use this parameter if the acronym is **not** at offset zero and BASELBL is not specified. Value may be a decimal constant, hexadecimal constant, or absolute expression.

PREFIX = *value*

specifies the number of characters to be removed from the front of a field name to produce the field label. The field name is defined by the NAME parameter of the BLSQMFLD macro. Value must be an integer constant (0 - 8, inclusive). When PREFIX=8 is specified, the fields will have no label. mode. If not specified, the default is PREFIX=3. PREFIX may be re-specified on a succeeding BLSQMFLD macro.

OFFSETS=PRINT
OFFSETS=NOPRINT

specifies whether or not the field offset information should be printed at the beginning of each output line of the formatted control block. PRINT specifies that offset information should be included on the formatted line; NOPRINT causes the offset information to be suppressed. If this parameter is not specified, a default of PRINT is used.

STRTCOL = *value*

specifies a left margin for each line of the formatted control block. Value may be a decimal constant, a hexadecimal constant, or an absolute expression. If not specified, or specified as zero, the format model processor uses the value specified by IPCS or printdump.

LBLSPC = *value*

specifies the spacing between label fields in the formatted output. Value may be a decimal constant, hexadecimal constant, or an absolute expression. If not specified, or specified as zero, this indicates to the format model processor that the value specified by IPCS, SNAP, or PRDMP should be used. This value is initially set to 20.

Note: If *value* is 18, the output is condensed.

HEADER = *name*

specifies the heading that will precede the formatted control block. Name may be any one to eight byte character string that contains no blanks. If HEADER is omitted, the ACRONYM value is used for the heading. If neither the ACRONYM parameter nor the HEADER parameter is specified, the formatted control block will not have a heading.

BLSQMFLD - Specifying a Control Block Format Field

The BLSQMFLD macro instruction is used to identify the fields within the dumped control block that are to be formatted. A BLSQMFLD macro must be coded for each requested field that will be formatted.

The BLSQMDEF and BLSQMFLD macro instructions work together to create a dump formatting model for a control block, the model has the following structure:

- One BLSQMDEF macro instruction to begin the model definition.
- At least one BLSQMFLD macro instruction to define the attributes of a desired control block field.
- One BLSQMDEF macro instruction to end the model definition.

The order of the BLSQMFLD statements in the formatting model determines the order the fields are printed in the dump. No object code producing assembler statements other than the BLSQMFLD macro instruction should be placed between the BLSQMDEF macro instructions that delimit the start and end of the model definition.

Through the implementation of BLSQMDEF and BLSQMFLD, users of IPCS, PRDMP, and SNAP can control their dump output within user-written formatting routines. The BLSQSHDR macro instruction, which associates text strings with dumped data fields, can be used to clarify the dump for the user. For additional information, refer to *MVS/XA Interactive Problem Control System User's Guide and Reference.*

The BLSQMFLD macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede BLSQMFLD.
BLSQMFLD	
b	One or more blanks must follow BLSQMFLD.
NAME=*label* NAME=*	*label:* symbol.
,SHDR=*addr*	*addr:* A-type address. **Note:** If SHDR is specified, only CALLRTN, NEWLINE, NOSPLIT, and VIEW are allowed.
,OFF=*value*	*value:* decimal constant, hexadecimal constant, or absolute value. **Note:** OFF is required if BASELBL is not specified on the BLSQMDEF macro or if NAME=* is specified on the BLSQMFLD macro.
,LEN=*value*	*value:* decimal constant, hexadecimal constant, or absolute expression. **Note:** LEN is required if name parameter label is unresolved.
,VIEW=*(list)* ,VIEW=*value*	*(list):* integers between 1 and 16, inclusive. *value:* decimal constant, hexadecimal constant, or absolute value. **Default:** VIEW=X'0200'
,ARRAY=*((DL1,DU1),(DL2,DU2))*	*DL1,DU1,DL2,DU2:* decimal constants, hexadecimal constants, or absolute values.
,ARRAY=*value*	*value:* decimal constant, hexadecimal constant, or absolute value.
,ARRAY=*	**Note:** LEN and OFF are ignored when you code any specification of ARRAY= other than ARRAY=END.
,ARRAY=END	END terminates an array definition.
,DTYPE=HEX ,DTYPE=EBCDIC	**Default:** DTYPE=HEX
,NEWLINE	
,NOLABEL	
,CALLRTN	
,PREFIX=*value*	*value:* integers between 0 and 8 **Note:** If omitted, *value* specified in the last preceding BLSQMDEF or BLSQMFLD macro is used.
,NOSPLIT	
,NUMDEC	**Default:** Hexadecimal.
,NOCOLNM	**Default:** Number the columns.
,STRTCOL=*value*	*value:* decimal constant, hexadecimal constant, or absolute value. **Default:** value specified by SNAP, IPCS, or PRDMP.
,COLNUM=*value*	*value:* decimal constant, hexadecimal constant, or absolute value. **Default:** A value is calculated.
,COLSEP=*value*	*value:* decimal constant, hexadecimal constant, or absolute value. **Default:** A value is calculated.
,ITEMSEP=*value*	*value:* decimal constant, hexadecimal constant, or absolute value. **Default:** A value is calculated.
,ORDER=(1,2) ,ORDER=(2,1)	**Default:** ORDER=(1,2)
,HEXONLY	

The parameters are explained as follows:

NAME = *label*
NAME = *

specifies the name of the control block field described by this BLSQMFLD macro. If BASELBL is specified on the BLSQMDEF macro, the NAME label will be used with the BASELBL label to calculate the offset of this field from the start of the control block. If BASELBL was not specified on the BLSQMDEF macro, then OFF becomes required on the BLSQMFLD macro.

A single asterisk specifies an unnamed, reserved field. Use of the single asterisk for the name of a control block field requires that the OFF and LEN parameters be specified. The dump formatter replaces the asterisk with a "reserved" label.

SHDR = *addr*

specifies the address of a character string used as a subheading in the control block format. The address must be valid in an assembler A-type DC instruction. This parameter should point to a one-byte length field followed by the actual heading character string. The length byte indicates the length of the heading string only and should not include the length of the length byte.

If this parameter is specified, only CALLRTN, NEWLINE, NOSPLIT, and VIEW can be specified. Other parameters will be ignored.

OFF = *value*

specifies the offset of this field from the beginning of the control block. The value may be a decimal constant, a hexadecimal constant, or an absolute expression. If this parameter is specified, the value defined overrides the default field offset generated by the NAME label on this macro and the BASELBL label on the BLSQMDEF macro.

OFF is ignored if you code any specification of ARRAY= other than ARRAY=END.

This parameter is required if the BASELBL parameter is not specified on the BLSQMDEF macro or if NAME=* is specified on the BLSQMFLD macro.

LEN = *value*

specifies the length of the control block field. The value is a decimal constant, hexadecimal constant, or absolute expression that defines the length of the control block field. This parameter is required if no data constants with a label exist in the assembly program as defined by the NAME parameter, or if use of the assembler length attribute would not result in a correct length determination for the data constant representing the field.

LEN is ignored if you code any specification of ARRAY= other than ARRAY=END.

An assembly error occurs if LEN is not specified and there is no assembler statement with a label matching the one specified by NAME.

VIEW=*(list)*
VIEW=*value*

specifies up to sixteen different views of the control block fields. Any combination of one to sixteen view attributes can be specified for each field. The last four bits of the view pattern of 16 are reserved for the control program. The caller of the dump formatter provides a view pattern defining those views to be formatted.

When an attribute in the view pattern supplied by the dump formatter's caller (in ADPLPFMT) matches an attribute in the field view pattern, the field is selected for formatting.

The list is an unordered list of attributes; each attribute can be a decimal integer between 1 and 16, inclusive (as in VIEW=1,2,...,16), binary constant (as in VIEW=B'0010'), or hexadecimal constant (as in VIEW=X'0080').

The following chart illustrates the view parameter's control block field options provided through the specification of a 4-digit hexadecimal number. Any combination of the view fields listed may be specified.

Hexadecimal Code	User-defined fields to be displayed
x'8000'	keyfield
x'4000'	summary field
x'2000'	register save area
x'1000'	linkage field
x'0800'	error fields
x'0400'	hexadecimal dump
x'0200'	all non-reserved fields
x'0100'	reserved fields
x'0080'	static array
x'0040'	dynamic array
x'0020'	input field
x'0010'	output field

If this parameter is not specified, the default value of VIEW=X'0200' is used. See *IPCS User's Guide and Reference* for more information about ADPLPFMT.

ARRAY=*((DL1,DU1),(DL2,DU2))*
ARRAY=*value*
ARRAY=*
ARRAY=END

specifies that the succeeding BLSQMFLD statements define a set of fields that are repeated in the control block.

Using the ARRAY parameter on the BLSQMFLD macro indicates that this particular BLSQMFLD macro instruction is the beginning or the end of an array definition.

The LEN and OFF parameters are ignored when you code any specification of ARRAY= other than ARRAY=END.

The VIEW specified applies to all fields within the array; therefore, the VIEW specified on the BLSQMFLD macro that starts an array should be the composite of the VIEW on all fields within the array.

If ARRAY=*((DL1,DU1),(DL2,DU2))* is coded, a two dimensional array is specified. *DL1* is the lower limit of the first dimension, *DU1* is the upper limit of the first dimension, and similarly for *DL2* and *DU2* for the second dimension. If a lower limit for a dimension is not specified, a default of 1 is provided. There is no default for the upper limit of a dimension. However, an asterisk (*) may be coded for either the upper limit or lower limit of the dimension to indicate that the dimension is to be provided by the calling program at execution time.

Notes:

1. *The correspondence of a dimension to either row or column is determined by the ORDER keyword.*

2. *If the array is larger than 65,535 bytes, the calling program must process the array in sections. The formatter will equate the lower limit for each dimension to the value one for the purpose of addressing the array entries in a buffer, but will use the specified values for the purpose of numbering rows and columns in the formatted output.*

If ARRAY=*value* is coded, a one dimensional array (list) is specified. *Value* defines how many array entries are contained in the control block.

If ARRAY=* is coded, the number of entries in the one-dimensional array (list) is to be be provided by the calling program at execution time.

If ARRAY=END is coded, the array definition is terminated.

NEWLINE
specifies that this field must start on a new line of formatted output.

DTYPE=HEX
DTYPE=EBCDIC
specifies the type of data contained in the area to be dumped. DTYPE=HEX indicates that the area to be dumped contains four-bit hexadecimal digits. DTYPE=EBCDIC indicates that the area to be dumped contains eight-bit EBCDIC characters. When you specify DTYPE=HEX, the dumped area includes the actual hexadecimal digits in the range 0-F, plus any EBCDIC characters that are equivalent to 2-digit combinations of those digits. The equivalent EBCDIC appears within vertical bars. When you specify DTYPE=EBCDIC, the dumped area includes only the EBCDIC characters, with nothing between the vertical bars.

NOLABEL
specifies that the field label is not to be printed. NAME is still required for offset calculation.

CALLRTN
specifies that the dump formatter calls the output line processing exit after the output line containing this field has been formatted but before it is printed. The output line processing exit entry point address is specified by the caller in the parameter list when the dump formatter is invoked.

PREFIX=*value*
specifies how many characters are to be removed from the front of a field name to produce the field label. The field name is defined by the NAME parameter. Value must be an integer constant greater than or equal to zero and less than or equal to eight. If PREFIX is omitted from the current BLSQMFLD macro, the value specified on the last

preceding BLSQMFLD or BLSQMDEF macro is used. The BLSQMDEF macro used to start a model definition may be used to set the value of PREFIX.

NOSPLIT
specifies that the dump formatter attempts to print all the field data on the same output line. If the data does not fit on the current output line but fits on a single output line, the dump formatter skips to a new line prior to printing this data field.

NUMDEC
specifies that the columns and rows of a two-dimensional array be numbered in decimal. The default is hexadecimal.

NOCOLNM
specifies that column numbers (headers) of a two-dimensional array be suppressed. The default is to number the columns. (The NUMDEC parameter controls the numbering system used for numbering the columns.)

STRTCOL = *value*
specifies the left margin of the formatted output. *Value* indicates the number of blanks before the first character. STRTCOL applies only to two-dimensional arrays. This specification overrides the value defined by the STRTCOL keyword in the BLSQMDEF macro, or by the host (IPCS, SNAP, or PRDMP), for the duration of displaying the array. If not specified, a default of zero is provided and the formatter will use the value specified by the host.

COLNUM = *value*
specifies the number of columns of a two dimensional array that are to be displayed as a group. If not specified, or if the specified number of columns will not fit in the currently available print buffer, the formatter will calculate a value consistent with, and not exceeding, the maximum line length specified by IPCS, SNAP, or PRDMP.

COLSEP = *value*
specifies the number of blanks to be placed between the columns of a two-dimensional array. The default is zero, and the formatter uses a calculated value.

ITEMSEP = *value*
specifies the number of blanks to be placed between items within an array entry. An array entry may be a structure, and each element of the structure is referred to as an "item". If the array entry is a single item, *value* will be ignored. If ITEMSEP is not specified, a default of zero is provided and the formatter will use a calculated value when needed.

ORDER = (1,2)
ORDER = (2,1)
specifies the order in which the data of a two-dimensional array are to be processed. If ORDER = (1,2) is specified, the data is processed in consecutive rows. If ORDER = (2,1) is specified, the data is processed in consecutive columns. The default is ORDER = (1,2).

HEXONLY
specifies that the data is to be displayed in hex only. If you omit HEXONLY, the data is displayed in both hex and EBCDIC, on the same line, with vertical bars bounding the EBCDIC portion of the display. HEXONLY is valid only if the view parameter specifies X'0400', which requests a hexadecimal dump.

Example 1

Operation: Code the macro instructions that will establish a control block formatting model to be used by the dump formatter to format functional recovery routines (FRRs).

```
IEAVTRP3 CSECT
         BLSQMDEF CBLEN=X'0320',MAINTLV=HBB2102,PREFIX=4,OFFSETS=PRINT,X
               HEADER=FRRS
         BLSQMFLD NAME=FRRSEMP,OFF=X'0000',LEN=4,VIEW=X'0202'
         BLSQMFLD NAME=FRRSLAST,OFF=X'0004',LEN=4,VIEW=X'0202'
         BLSQMFLD NAME=FRRSELEN,OFF=X'0008',LEN=4,VIEW=X'0202'
         BLSQMFLD NAME=FRRSCURR,OFF=X'000C',LEN=4,VIEW=X'0200'
         BLSQMFLD NAME=FRRSRSA,OFF=X'0010',LEN=24,VIEW=X'0200'
         BLSQMFLD SHDR=RTM1WA,VIEW=X'0200',NEWLINE
         BLSQMFLD SHDR=BLANK,VIEW=X'0200',NEWLINE
         BLSQMFLD SHDR=ENTEXT,VIEW=X'0200',NEWLINE
         BLSQMFLD SHDR=BLANK,VIEW=X'0200',NEWLINE
         BLSQMFLD NAME=FRRSXSTK,VIEW=X'0200',ARRAY=16,NOLABEL
         BLSQMFLD NAME=FRRSKM,OFF=X'00A0',LEN=2,VIEW=X'0200',NEWLINE
         BLSQMFLD NAME=FRRSSAS,OFF=X'00A2',LEN=2,VIEW=X'0200'
         BLSQMFLD NAME=FRRSAX,OFF=X'00A4',LEN=2,VIEW=X'0200'
         BLSQMFLD NAME=FRRSPAS,OFF=X'00A6',LEN=2,VIEW=X'0200',ARRAY=END
         BLSQMFLD SHDR=BLANK,VIEW=X'0200',NEWLINE
         BLSQMFLD SHDR=ENTS,VIEW=X'0200',NEWLINE
         BLSQMFLD SHDR=BLANK,VIEW=X'0200',NEWLINE
         BLSQMFLD NAME=FRRSENTS,VIEW=X'0200',ARRAY=16,NOLABEL
         BLSQMFLD NAME=FRRSFRRA,OFF=X'0120',LEN=4,VIEW=X'0200',NEWLINE
         BLSQMFLD NAME=FRRSFLGS,OFF=X'0124',LEN=4,VIEW=X'0200'
         BLSQMFLD NAME=FRRSPARM,OFF=X'0128',LEN=24,VIEW=X'0200',       X
               ARRAY=END
         BLSQMDEF END
BLANK    BLSQSHDR ' '
ENTEXT   BLSQSHDR 'FRR ENTRY EXTENSIONS'
ENTS     BLSQSHDR 'FRR ENTRIES'
RTM1WA   BLSQSHDR 'RTM1 WORK AREA FOLLOWS FRR ENTRIES'
         END
```

Example 2

Operation: Code the macro instructions that will establish a control block formatting model to be used by the dump formatter to format a STAE control block (SCB).

```
IEAVTRP4 CSECT
         BLSQMDEF CBLEN=X'0018',MAINTLV=JBB2125,PREFIX=3,OFFSETS=PRINT,X
               HEADER=SCB
         BLSQMFLD NAME=SCBCHAIN,OFF=X'0000',LEN=4,VIEW=X'0200'
         BLSQMFLD NAME=SCBEXIT,OFF=X'0004',LEN=4,VIEW=X'0200'
         BLSQMFLD NAME=SCBFLGS1,OFF=X'0008',LEN=1,VIEW=X'0200'
         BLSQMFLD NAME=SCBPARMA,OFF=X'0009',LEN=3,VIEW=X'0200'
         BLSQMFLD NAME=SCBFLGS2,OFF=X'000C',LEN=1,VIEW=X'0200'
         BLSQMFLD NAME=SCBOWNRA,OFF=X'000D',LEN=3,VIEW=X'0200'
         BLSQMFLD NAME=SCBFLGS3,OFF=X'0010',LEN=1,VIEW=X'0200'
         BLSQMFLD NAME=SCBPKEY,OFF=X'0011',LEN=1,VIEW=X'0200'
         BLSQMFLD NAME=SCBID,OFF=X'0012',LEN=1,VIEW=X'0200'
         BLSQMFLD NAME=SCBRSVRE,OFF=X'0013',LEN=1,VIEW=X'0200'
         BLSQMFLD NAME=SCBXPTR,OFF=X'0014',LEN=4,VIEW=X'0200'
         BLSQMFLD NAME=*,OFF=X'0000',LEN=X'0018',VIEW=X'0400',NOLABEL
         BLSQMDEF END
         End
```

Example 3

Operation: Define the format of a very simple control block. Note that this could be done by using a macro-invocation.

```
MYBLK    DSECT ,                  My simplest control block ever
MYBLKABC DC    C'ABC'             Identifier
MYBLKDEF DC    X'00'              Flags
MYBLKD80 EQU   X'80'              1st flag bit
MYBLKD40 EQU   X'40'              2nd flag bit
MYBLKGHI DC    V(MYENTRY)         Address of my program
MYBLKEND EQU   *                  End of my control block
```

Define enough storage to get the block displayed. Note that no ENTRY statement is required for access to CBMODEL1 from other CSECTs since CBMODEL1 lies at the origin of the CSECT.

```
         TITLE 'CBMODEL1--Basic Control Block Model'
CBMODEL  CSECT ,                  Start definition of simple model
CBMODEL1 BLSQMDEF BASELBL=MYBLK,CBLEN=MYBLKEND-MYBLK,PREFIX=5
         BLSQMFLD NAME=MYBLKABC
         BLSQMFLD NAME=MYBLKDEF
         BLSQMFLD NAME=MYBLKGHI
         BLSQMDEF END             End definition of simple model
```

Add acronym checking, the display of the acronym in EBCDIC, and descriptive header for the display in the dump.

```
         TITLE 'CBMODEL2--More Elaborate Than 1st Model'
         ENTRY CBMODEL2           Permit access from other CSECTs
CBMODEL2 BLSQMDEF BASELBL=MYBLK,CBLEN=MYBLKEND-MYBLK,PREFIX=5,    X
               ACRONYM=ABC,ACROLBL=MYBLKABC, Acronym field data
               HEADER=MYBLOCK     Heading for block in dump
         BLSQMFLD NAME=MYBLKABC,DTYPE=EBCDIC Show it as EBCDIC data
         BLSQMFLD NAME=MYBLKDEF
         BLSQMFLD NAME=MYBLKGHI
         BLSQMDEF END             End definition of alternate model
         END   CBMODEL1           End definition of formatting model
```

Example 4

Operation: Assume the data is stored in this sequence:

```
00010001
00010002
00010003
00010004
00020001
00020002
00020003
00020004
00030001
00030002
00030003
00030004
   .
   .
   .
00090001
00090002
00090003
00090004
00100001
00100002
00100003
00100004
```

And you want the data to be formatted as follows:

```
    ---01--- ---02--- ---03--- ---04---
    ARRENTRY ARRENTRY ARRENTRY ARRENTRY
    -------- -------- -------- --------
001 00010001 00010002 00010003 00010004
002 00020001 00020002 00020003 00020004
003 00030001 00030002 00030003 00030004
004 00040001 00040002 00040003 00040004
005 00050001 00050002 00050003 00050004
006 00060001 00060002 00060003 00060004
007 00070001 00070002 00070003 00070004
008 00080001 00080002 00080003 00080004
009 00090001 00090002 00090003 00090004
010 00100001 00100002 00100003 00100004
```

Therefore, code the macro instruction that will create a formatting model to do the following:

Number rows 1 through 10.
Number columns 1 through 4.
Use the decimal numbering system for numbering rows and columns.
Place data in to the array row by row.
Put one blank between each column.
Display 4 columns in each group.
Start printing in the second column from the left margin.
View all non-reserved fields.
Print the field label ARRENTRY.

One way to code the macro:

```
BLSQMFLD NAME=ARRAYX,ARRAY=((1,10),(1,4)),VIEW=X'0200',    X
      STRTCOL=1,COLSEP=1,COLNUM=4,NUMDEC,NOLABEL
BLSQMFLD NAME=ARRENTRY,OFF=0,LEN=4,ARRAY=END,VIEW=X'0200'
```

Example 5

Operation: Assume the data is stored in this sequence:

```
00010001
00010002
00010003
00010004
00020001
00020002
00020003
00020004
00030001
00030002
00030003
00030004
   .
   .
   .
00090001
00090002
00090003
00090004
00100001
00100002
00100003
00100004
```

And you want the data to be formatted as follows:

```
    ---05--- ---06--- ---07--- ---08--- ---09---
    ARRENTRY ARRENTRY ARRENTRY ARRENTRY ARRENTRY
    -------- -------- -------- -------- --------
000 00010001 00020001 00030001 00040001 00050001
001 00010002 00020002 00030002 00040002 00050002
002 00010003 00020003 00030003 00040003 00050003
003 00010004 00020004 00030004 00040004 00050004

    ---0A--- ---0B--- ---0C--- ---0D--- ---0E---
    ARRENTRY ARRENTRY ARRENTRY ARRENTRY ARRENTRY
    -------- -------- -------- -------- --------
000 00060001 00070001 00080001 00090001 00100001
001 00060002 00070002 00080002 00090002 00100002
002 00060003 00070003 00080003 00090003 00100003
003 00060004 00070004 00080004 00090004 00100004
```

Therefore, code the macro instruction that will create a formatting model to do the following:

Number rows 0 through 3.
Number columns 5 through 14.
Use the hexadecimal numbering system for numbering rows and columns.
Put two blanks between each column.
Display 5 columns in each group.
Start printing in the fourth column from the left margin.
View all non-reserved fields.
Print the field label ARRENTRY.

One way to code the macro:

```
BLSQMFLD NAME=ARRAYX,ARRAY=((5,14),(0,3)),VIEW=X'0200',     X
      STRTCOL=3,COLSEP=2,COLNUM=5,NOLABEL,ORDER=(2,1)
BLSQMFLD NAME=ARRENTRY,OFF=0,LEN=4,ARRAY=END,VIEW=X'0200'
```

BLSQSHDR - Generate Model Subheader

The BLSQSHDR macro instruction lets you define a text string, called a subheader, and associate it with a particular data field in a dump format. Whenever the dump occurs, the text string appears in the dump as an aid in spotting the associated data field.

BLSQSHDR, with its text string, should be placed after the end of the format model definition. You create a format model definition by coding two BLSQMDEF macros, one at the beginning of the definition and another at the end. The BLSQMFLD macros, which define the data fields of the format model, are included between these two BLSQMDEF macros. The SHDR fields of the included BLSQMFLD macros reference text strings (subheaders) that you have placed after the end of the model definition. The order of the macros is:

```
BLSQMDEF
BLSQMFLD
    .
    .
    .
BLSQMFLD
BLSQMDEF
BLSQSHDR
```

Thus, each BLSQSHDR macro placed after the end of the model must have a label that can be referenced by the BLSQMFLD macros within the model. The text string of the BLSQSHDR macro is enclosed in single quotation marks. L(x) may also be coded if the length of the string is different than the length of the enclosed text string.

The BLSQSHDR macro instruction is written as follows:

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

L(*x*)	*x*: Length of subheader - if other than length of actual text
'text'	*text*: text of subheader

L(*x*)
specifies the length of the subheader. Only necessary if the length is to be different from the length of the enclosed text string.

Example

```
SHDR01  BLSQSHDR 'This is a subheader'
SHDR02  BLSQSHDR L(6)' '
```

BLSRESSY - Map IPCS Symbol Table Record

The BLSRESSY macro instruction maps the symbol table record that a user-written exit routine (operating under IPCS) passes to the get symbol and equate symbol services.

With the BLSRESSY macro instruction, users of the get symbol and equate symbol services can retrieve definitions described in the IPCS symbol table and create definitions for later use by the IPCS user or by other routines. For information about the get symbol and equate symbol services, see *MVS/XA Interactive Problem Control System User's Guide and Reference.*

The BLSRESSY macro instruction is written as follows:

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

DSECT=YES DSECT=NO	**Default:** DSECT=YES

NOTE: Users must supply a label (*name*), and start it in column 1 of the BLSRESSY macro instruction. When the BLSRESSY macro is executed, the label becomes the record name and the prefix to the name of each field in the record.

The parameters are explained as followed:

DSECT=YES
DSECT=NO
specifies whether the record mapped by BLSRESSY is to be mapped as a DSECT (YES) or not (NO).

Example

Operation: Map the IPCS symbol table record but not as a DSECT.

```
ESSY     BLSRESSY DSECT=NO
```

CALLDISP - Force Dispatcher Entry

If you are executing in 31-bit addressing mode, you must use the MVS/XA version of this macro instruction.

The CALLDISP macro instruction expands into an SVC or branch that results in the caller's status being saved in the current TCB/RB and 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.

When control returns to the caller:

- The cross memory mode is unchanged.
- Registers 14-1 are destroyed if FIXED=NO is specified; otherwise registers are unchanged.
- No locks are held.
- Control returns enabled.
- PCLINK status is saved and restored.

The CALLDISP macro instruction is written as follows:

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

BRANCH=NO BRANCH=YES	**Default:** BRANCH=NO
,FIXED=YES ,FIXED=NO	**Default:** (Available only if BRANCH=YES is coded) FIXED=YES
,FRRSTK=SAVE ,FRRSTK=NOSAVE	**Default:** (Available only if BRANCH=YES is coded) FRRSTK=NOSAVE

The parameters are explained as follows:

BRANCH=NO
BRANCH=YES
specifies whether the branch entry (BRANCH=YES) or the SVC entry (BRANCH=NO)to the dispatcher is to be used. BRANCH=YES is restricted to key 0 supervisor state callers. The default is BRANCH=NO. Routines that are unlocked, have no enabled unlocked task FRRs on the stack, and are in home mode can use BRANCH=NO. "Using the BRANCH=YES Option of the CALLDISP Macro Instruction" in Volume 1 lists requirements for routines that use BRANCH=YES.

,FIXED=YES
,FIXED=NO
specifies that the code invoking branch entry CALLDISP is in fixed storage (FIXED=YES) or in pageable storage (FIXED=NO). For FIXED=NO, registers 14-1 are altered.

,FRRSTK=SAVE
,FRRSTK=NOSAVE
specifies that the current FRR stack be saved and restored (FRRSTK=SAVE), if at least one of the FRRs is an enabled unlocked task (EUT) FRR, or not saved (FRRSTK=NOSAVE).

When FRRSTK=SAVE is specified:

- The caller must not hold any locks or an abend results.

  Note: For MVS/System Product Version 2 Release 1.3 Vector Facility Enhancement or MVS/System Product Version 2 Release 1.3 Availability Enhancement and later releases:

  - If any EUT FRRs exist, the current FRR stack is saved and the caller may hold either the LOCAL or CML lock. CALLDISP releases the lock before going to the dispatcher.

  - If no EUT FRR exists, the caller cannot hold any locks. Otherwise, an abend occurs.

- Asynchronous exits (IRBs and SIRBs) are not dispatched until all EUT FRRs have been deleted.

For more information, see "Suspension and Resumption of Request Blocks" in Volume 1 for an explanation of the CALLDISP function used with SUSPEND/RESUME processing.

Specifying FRRSTK=NOSAVE causes the FRR stack to be purged and the LOCAL or CML lock to be released before entering the dispatcher.

Note: If there are any EUT FRRs on the stack, the SVC interface to CALLDISP cannot be used; the BRANCH=YES option must be used.

Example 1

Operation: Pass control to another ready task.

```
CALLDISP
```

Example 2

Operation: A non-page-fixed task with an enabled, unlocked task FRR gives control to the dispatcher. When the task regains control, the contents of registers 14, 15, 0 and 1 will have changed.

```
CALLDISP  FIXED=NO,FRRSTK=SAVE,BRANCH=YES
```

CALLRTM - Call Recovery Termination Manager

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

Only key zero supervisor state routines can use CALLRTM. If the current address space is terminated (MEMTERM), control might or might not return to the caller before the MEMTERM takes effect. See "Invoking the Recovery Termination Manager" in Volume 1 for the complete recovery termination interface.

Except for the TCB, all input parameters to this macro instruction can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

The CALLRTM macro instruction is written as follows:

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

TYPE=ABTERM TYPE=MEMTERM	
,COMPCOD=*comp code*	*comp code:* symbol, decimal digit, or register (2) - (12).
,REASON=*code*	*code:* a symbol, decimal or hexadecimal number, or register (2) - (12).
,ASID=*asid*	*asid:* decimal digits 0-32,765 or register (2) - (15).
,TCB=*tcb addr*	*tcb addr:* 0, or register (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 as follows:

TYPE=ABTERM
TYPE=MEMTERM
specifies whether the services of the recovery termination manager are being directed towards task termination (ABTERM) or address space termination (MEMTERM). For MEMTERM, all recovery processing in the address space is skipped.

Unless ASID is also specified, TYPE=ABTERM is supported in home mode only. In a cross memory environment, if ASID is not specified, the TCB must reside in the home address space; if ASID is specified, the TCB must be in the same address space as the ASCB.

,COMPCOD=*compcode*
specifies the system 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.

,REASON=*code*
specifies additional information to supplement the completion code associated with an abnormal termination. The value range for the reason code is any 32-bit hexadecimal number or 31-bit decimal number. In all cases the result is hexadecimal.

If the reason code is explicitly specified using the REASON parameter, the hexadecimal representation of the code is passed to RTM in register 6 and a flag (X'04') is set in byte 0 in general register 1. If the REASON code is not specified, this flag is set to 0.

The reason code value is passed to recovery exits in the SDWACRC field of the SDWA. This value can be altered by the SETRP macro instruction. If altered, the altered value is sent to the next recovery exit.

,ASID=*asid*
specifies the address space identifier of the address space to be terminated (for MEMTERM) or the address space identifier of the address space containing the TCB of the task to be terminated (for ABTERM). If you omit this parameter or specify zero, the current address space is assumed. If you specify this parameter, you must supply an 18-word work area and pass its address in register 13.

Note: The contents of register 2 is destroyed if this parameter is used.

,TCB=*tcb addr*
specifies the TCB address of the task to be terminated. In a cross memory environment, if ASID is not specified, the TCB must reside in the home address space; if ASID is specified, the TCB must be in the same address space as the ASCB.

Note: The TCB resides in storage below 16 megabytes.

,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 are defined by the //SYSABEND, //SYSMDUMP, or //SYSUDUMP 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 can be created using the list form of the SNAP macro instruction, or a compatible list can be created. The system dump options specified by the CHNGDUMP operator command can add to or override this parameter list. All recovery routines entered for the failure can also add to the list of dump options. The TCB, DCB, and STRHDR options available on SNAP are ignored if they appear in the parameter list; the TCB used is for the task that received the ABEND and the DCB used is provided by the ABDUMP routine. If a //SYSABEND, //SYSMDUMP, or //SYSUDUMP DD statement is not provided, the DUMPOPT parameter is ignored.

Note: The contents of register 3 is destroyed if this parameter is used.

Register 15 contains one of the following return codes for TYPE=MEMTERM only:

Hexadecimal Code	Meaning
0	The MEMTERM request was processed successfully.
4	MEMTERM processing was not performed. The address space was marked as not suitable for MEMTERM processing. RTM writes an entry to SYS1.LOGREC if it rejected the MEMTERM request due to a damaged ASCB, if the address space must not be terminated, or if ASID exceeds ASVTMAX.

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 the TCB whose address is specified in register 8 for abnormal termination. The abnormal termination of this TCB takes place in the address space identified by the ASID specified in register 5, and has a completion code of 123.

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

CBPZDIAG - Build Diagnostic Stack Entry

The CBPZDIAG macro must be included in the unit information module (UIM) that an installation provides for any device that the MVS configuration program (MVSCP) does not support. See *SPL: System Modifications* for a complete description of coding a UIM.

The CBPZDIAG macro builds a diagnostic stack entry. The diagnostic stack entry contains debugging information that is placed in the system diagnostic work area (SDWA) if an ABEND occurs in the UIM. The diagnostic stack entry is contained within the UIM.

Note: A UIM must not establish an ESTAE to provide diagnostic information in the event that it ABENDs. Rather, it must:

1. *Specify the diagnostic information in a diagnostic stack entry, using the CBPZDIAG macro.*
2. *Use the CBPZPPDS macro to put the entry on the diagnostic stack in its entry logic.*
3. *Use the CBPZPPDS macro to remove the entry from the diagnostic stack in its exit logic.*

The ESTAE routine in the control routine for the MVS configuration program (CBPMVSCP) uses the information in the active diagnostic stack entry to fill in the SDWA. Also, the ESTAE routine builds a symptom string in the variable recording area (VRA) consisting of all the CSECT names in the entries on the diagnostic stack.

The CBPZDIAG macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede CBPZDIAG.
CBPZDIAG	
b	One or more blanks must follow CBPZDIAG.
MODNAME=*modname*	*modname:* CBPUCnnn n is a decimal digit.
CSECT=*csectname*	*csectname:* CBPUCnnn n is a decimal digit.
COMP=*id*	*id:* component identifier, 5 bytes long.
DESC=*text*	*text:* character string in quotes.
VRADATA=*label*	*label:* symbolic label
,RELATED=*value*	*value:* any valid macro keyword specification.

name
specifies the label on the diagnostic stack entry. The labels on the fields generated in the diagnostic stack entry will start with the same characters as **name** does. (If **name** exceeds four characters, only the first four characters will be used in building the labels on the generated fields.) This name is required.

MODNAME = *modname*
specifies the name of the load module that contains the diagnostic stack entry. If an ABEND occurs, this value will be placed in SDWA field SDWAMODN. The module name is eight characters long and is in the form of CBPUCnnn, where nnn is a decimal number from 001 to 256, inclusive, for customer-written UIMs. This parameter is required.

CSECT = *csectname*
specifies the name of the CSECT that contains the diagnostic stack entry. If an ABEND occurs, this value will be placed in SDWA field SDWACSCT. This parameter is optional. The default for this parameter is the assembler symbol, &SYSECT.

COMP = *id*
specifies the component identifier of the UIM. If an ABEND occurs, this value will be placed in SDWA field SDWACID. The component identifier should be five bytes long. This parameter is required.

DESC = *text*
specifies the UIM description, which should contain the unit names of the device(s) that the UIM supports. If an ABEND occurs, this value will be placed in SDWA field SDWASC. The UIM description can be a maximum of 23 bytes long. This parameter is required.

VRADATA = *label*
specifies the name of an array that contains the addresses of data to be placed in the VRA, if an ABEND occurs. The array contains the VRA keys and data lengths, in addition to the data addresses. This parameter is optional. If it is not specified, no specific control blocks or data areas for the UIM will be placed in the VRA. (On IODEVICE calls, the diagnostic stack entry for CBPICBBR, which is the routine that invokes UIMs on IODEVICE calls, causes the IODV to be placed in the VRA.)

Each entry in the VRA array contains eight bytes. The format of an entry is as follows:

Offset	Length	Function
0	2	Reserved, must be set to zero in all but the last entry in the array.
2	1	Key of VRA data, as specified in IHAVRA.
3	1	Length of VRA data.
4	4	Address of VRA data. If this field is set to zero, the ESTAE routine will skip this entry when moving data into the VRA. UIMs are permitted to dynamically update this field while the diagnostic entry is on the diagnostic stack.

The last entry in the VRA array must be set to X'FFFFFFFFFFFFFFFF'.

,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.

CBPZLOG - Log an MVS Configuration Program Message

The CBPZLOG macro can be used only in the unit information module (UIM) that an installation provides for any device that the MVS configuration program (MVSCP) does not support. See *SPL: System Modifications* for a complete description of coding a UIM.

The CBPZLOG macro is used to issue a message to the MVS configuration program log file. A UIM must have addressability to the CPVT when it issues the CBPZLOG macro. It must also invoke the CBPZLOGR mapping macro. (CBPZLOGR maps the parameter list that is built by the CBPZLOG macro.)

The CBPZLOG macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede CBPZLOG.
CBPZLOG	
b	One or more blanks must follow CBPZLOG.
MID=*id*	*id:* CBPnnnI. n is a decimal digit.
,SEV=*value* ,STMT=ITRHSNBR	*value:* LOGRINFO, LOGRWARN, LOGRERR, or LOGRTERM. must be coded as shown.
,TEXT=*label*	*label:* symbolic label.

name
specifies the label to be generated on the first instruction in the macro expansion. The name is optional.

MID=*id*
specifies the message identifier. The message identifier is seven characters long and is in the form of CBPnnnI, where nnn is a decimal number from 900 to 999 inclusive for user-written UIMs. This parameter is required.

SEV=*value*
specifies the message severity. The following severities are supported:

LOGRINFO
informational message. This message has no effect on MVS configuration processing or its return code.

LOGRWARN
warning message. This message has no effect on MVS configuration program processing but will cause a return code of 4 to be issued (unless a higher severity message is issued.)

LOGRERR
error message. This message will prevent the MVS configuration program from building any I/O configuration members, and will cause a return code of 8 to be issued (unless a higher severity message is issued.)

LOGRTERM
terminating message. This message causes the MVS configuration program to terminate its processing and issue a return code of 16. A UIM must never issue a terminating message.

This parameter, which is optional, defaults to LOGRERR.

Note: The equates LOGRINFO, LOGRWARN, LOGRERR and LOGRTERM are generated by the CBPZLOGR macro.

STMT=ITRHSNBR
specifies the number of the statement in the MVS configuration program input stream that the message refers to. Field ITRHSNBR in the internal text record header (mapped by CBPZITRH) contains the statement number. This parameter is optional. If it is omitted, no statement number will be associated with the message.

TEXT=*label*
specifies the label of the message text. This text contains up to 255 bytes of data. The length of the text is determined by the length attribute of this field. This parameter is required.

Note: The message service will compress multiple blanks in the text and will split the text across multiple lines if necessary.

CBPZPPDS - Push/Pop Diagnostic Stack Entry

The CBPZPPDS macro must be included in the unit information module (UIM) that an installation provides for any device that the MVS configuration program (MVSCP) does not support. See *SPL: System Modifications* for a complete description of coding a UIM.

The CBPZPPDS macro is used to push an entry on (put an entry on) or pop an entry from (remove an entry from) the diagnostic stack. A UIM must have addressability to the CPVT when it issues the CBPZPPDS macro. It must also invoke the CBPZDIAG macro to build the diagnostic stack entry that is to be pushed on or popped from the diagnostic stack.

The CBPZPPDS macro is written as follows:

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

PUSH POP	
DIAG=*label*	*label:* symbolic label.
,RELATED=*value*	*value:* any valid macro keyword specification.

name
specifies the name on the first instruction in the macro expansion. The name is optional.

PUSH
specifies that the designated diagnostic entry is to be put on the diagnostic stack. Either PUSH or POP must be specified.

POP
specifies that the designated diagnostic entry is to be removed from the diagnostic stack. Either PUSH or POP must be specified.

DIAG=*label*
identifies the diagnostic entry. This name must be specified on the label field of the CBPZDIAG macro invocation.

,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.

See the CBPZDIAG macro description for more information about diagnostic entries.

CHANGKEY - Change Virtual Storage Protection Key

The CHANGKEY macro instruction changes the protection key and fetch protection status of one or more pages of virtual storage. The CHANGKEY function is available only for use by system components that execute in supervisor state and key zero. Callers can be enabled or disabled and cannot hold any lock that would prevent RSM from obtaining any RSM lock.

The CHANGKEY function is valid for virtual storage obtained by GETMAIN in page multiples from problem program subpools. Callers must provide an 18-word save area and place the address of the save area in register 13. If the caller is disabled, the save area must be in fixed storage.

The CHANGKEY macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede CHANGKEY.
CHANGKEY	
b	One or more blanks must follow CHANGKEY.
R,BA=*page addr*,EA=*page addr* L,LISTAD=*list addr*	*page addr:* A-type address or register (1) - (12). **Note:** The R-type macro expansion alters the contents of register 2. EA should not be specified as (1). *list addr:* A-type address or register (1) - (12).
,KEY=*stor key*	*stor key:* Decimal digit 1-15 or register (0) or register (3) - (12).
,BRANCH=YES	Required.

The parameters are explained as follows:

R,BA=*page addr*
EA=*page addr*
L,LISTAD=*list addr*
specifies the type of CHANGKEY request:

- R indicates a request to change the key of a single area of virtual storage.
- L indicates a request to change the key of one or more areas of virtual storage.
- BA specifies the address of the first byte of the first page of the virtual storage area whose key is to be changed.
- EA specifies the address of the first byte of the last page of the virtual storage area whose key is to be changed.

Notes:

1. *BA≤EA*

2. *BA, EA, and LISTAD are expected to be 31-bit addresses, regardless of the addressing mode of the issuer of the macro.*

LISTAD specifies the address of the first double-word of a variable length parameter list in fixed storage. The first word of each element is defined as BA above and the second word of each element as EA above. If the high-order bit of the second word is one that element is the last element in the parameter list.

,KEY=*stor key*
specifies the new storage key and fetch protection status for the virtual storage areas specified. If the *stor key* specification is a decimal digit, then the supervisor assumes the user wants fetch protection. If the user does not want fetch protection, he should specify the protection key he wants in bits 24-27 of a register and leave bit 28 at zero to indicate that he doesn't want fetch protection.

,BRANCH=YES
The only entry available into the CHANGKEY service routine is branch entry.

Note: The requestor must have addressability to the CVT.

Upon completion of the CHANGKEY macro instruction, register 15 contains a zero return code. If a caller requested that the key be changed to key 0, the caller is abended with a code X'08F'.

Example 1

Operation: Change the storage key and ensure fetch protection of a single page of virtual storage addressed by register 5.

```
CHANGKEY R,BA=(REG5),EA=(REG5),KEY=8,BRANCH=YES
```

Example 2

Operation: Change the storage key and ensure fetch protection of two noncontiguous pages of virtual storage addressed by PAGE1 and PAGE2 respectively.

```
CHANGKEY L,LISTAD=PLIST,KEY=10,BRANCH=YES
         .
         .
         .
PLIST    DC 2A(PAGE1)     FIRST ELEMENT IN LIST
         DC A(PAGE2)      BA PART OF SECOND ELEMENT
         DC AL1(X'80')    INDICATES LAST ELEMENT IN LIST
         DC AL3(PAGE2)    EA PART OF SECOND ELEMENT
```

CIRB - Create Interruption Request Block

The CIRB 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 scheduling the execution of user exit routines.

Branch Entry Interface

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

- The caller must be in supervisor state, key zero, and own the LOCAL lock and no locks above the SALLOC lock 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 is used.
- The caller must include the CVT mapping macro.
- Upon return, register 1 contains the address of the created IRB, registers 0, and 2-14 are unchanged, and register 15 is unpredictable.
- Control is returned in supervisor state, key zero, with the same locks held as on entry.

Note: The IRB address is returned in register 1.

The CIRB macro instruction is written as follows:

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

EP=*entry point addr*	*entry point 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 specified above.
,AMODE=CALLER ,AMODE=DEFINED	**Default:** AMODE=CALLER

The parameters are explained as follows:

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

,KEY=PP
,KEY=SUPR
specifies whether the asynchronous exit routine operates 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 executes in problem program (PP) or supervisor (SUPR) mode.

,SVAREA=NO
,SVAREA=YES
specifies whether to obtain a 72-byte register save area 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) is to be freed by EXIT.

Note: If the STAB parameter is omitted from the CIRB macro instruction, the IRB remains 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 IQEs. The first four bytes of the work area that is obtained contains the address of the next available IQE (RBNEXAV field). 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 is (YES) or is not (NO) returned to the available queue when the asynchronous exit terminates.

,AMODE=CALLER
,AMODE=DEFINED
specifies the addressing mode in which the exit routine is to be given control.

If CALLER is specified, the exit routine receives control in the same addressing mode as the caller.

If DEFINED is specified, the addressing mode of the exit routine is pointer defined. This means that the addressing mode is determined by the setting of the high order bit of the *entry point address* for the exit routine. If the bit is set, the addressing mode is 31-bit; if the bit is not set, the addressing mode is 24-bit.

Example 1

Operation: Create an IRB to be used in scheduling an asynchronous exit. The exit is scheduled via the IQE interface to stage 2 exit effector, and receives control in the supervisor state. The IRB is to be freed when it terminates. The exit receives 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 is used to schedule the routine. The exit gets control at the RQETEST label.

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

CPOOL - Perform Cell Pool Services

The CPOOL macro instruction creates a cell pool, obtains or returns a cell to the cell pool, or deletes the previously built cell pool, according to the function requested.

The CPOOL macro instruction is also described in *Supervisor Services and Macro Instructions* with the exception of the KEY, TCB, and LINKAGE=BRANCH parameters. LINKAGE=BRANCH can be used only by callers in supervisor state and key 0. TCB and KEY can be used only by supervisor state, key 0-7, or APF-authorized callers. Problem programs cannot create cell pools in subpools greater than 127. In order to create a cell pool in a subpool greater than 127, the user must be in system key, supervisor state, or be APF-authorized. On entry to this macro, users who specify the parameters: BUILD, DELETE, or REGS=SAVE must pass the address of a 72-byte save area in register 13.

The caller's secondary ASID is preserved when a PC instruction is issued; however, the caller cannot be in secondary addressing mode when issuing this macro instruction.

Except for the TCB, all input parameters to this macro instruction can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

The CPOOL macro instruction is written as follows:

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

BUILD GET FREE DELETE	
,UNCOND ,U ,COND ,C	**Default:** UNCOND **Note:** This parameter can be specified only with the GET keyword.
,PCELLCT=*primary cell count*	*cell count:* symbol, decimal digit, or register (0), (2) - (12). **Note:** This parameter can be specified only with the BUILD keyword.
,SCELLCT=*secondary cell count*	**Default:** PCELLCT **Note:** This parameter can be specified only with the BUILD keyword.
,CSIZE=*cell size*	*cell size:* symbol, decimal digit, or register (0), (2) - (12). **Note:** This parameter can be specified only with the BUILD keyword.
,SP=*subpool number*	*subpool number:* symbol, decimal digit, or register (0), (2) - (12). **Default:** SP=0 **Note:** This parameter can be specified only with the BUILD keyword.
,LOC=BELOW ,LOC=(BELOW,ANY) ,LOC=ANY ,LOC=RES ,LOC=(RES,ANY)	**Default:** LOC=RES **Note:** This parameter can be specified only with the BUILD keyword.
,CPID=*pool id*	*pool id:* RX-type address or register (0), (2) - (12). **Note:** This parameter must be specified with the GET, FREE, and DELETE keywords but is optional with the BUILD keyword.
,CELL=*cell addr*	*cell addr:* RX-type address or register (0), (2) - (12). **Note:** This parameter is required with the FREE keyword, is optional with the GET keyword, and cannot be specified with the BUILD and DELETE keywords.
,KEY=*key number*	*key number:* decimal digits 0-15 or register (0), (2) - (12). **Note:** This parameter can be specified only with the BUILD keyword.
,TCB=*tcb addr*	*tcb addr:* RX-type address or register (0), (2) - (12). **Default:** TCB address in PSATOLD. **Note:** This parameter can be specified only with the BUILD keyword.
,HDR=*hdr*	*hdr:* character string enclosed in single quotes, RX-type address, or register (0), (2) - (12). **Default:** 'CPOOL CELL POOL' **Note:** This parameter can be specified only with the BUILD keyword.
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	**Default:** LINKAGE=SYSTEM **Note:** This parameter cannot be specified with FREE or GET conditionally.
,REGS=SAVE ,REGS=USE	**Default:** REGS=SAVE **Note:** This parameter can be specified only with the GET or FREE keywords.

The parameters are explained as follows:

BUILD
GET
FREE
DELETE
specifies the cell pool service to be performed.

BUILD creates a cell pool in a specified subpool by allocating storage and chaining the cells together.

GET attempts to obtain a cell from the previously built cell pool. This request can be conditional or unconditional as described under the UNCOND/COND keyword.

FREE returns a cell to the cell pool.

DELETE deletes a previously built cell pool and frees storage for the initial extent, all secondary extents, and all pool control blocks.

,UNCOND
,U
,COND
,C
when used with GET specifies whether the request for a cell is conditional or unconditional. If COND or C is specified and the cell pool is empty, the CPOOL service routine returns to the caller without a cell and places a zero in the cell address. If UNCOND or U is specified and the cell pool is empty, the CPOOL service routine extends the pool in order to obtain a cell for the caller.

,PCELLCT = *primary cell count*
specifies the number of cells expected to be needed in the initial extent of the cell pool. The CPOOL service module uses PCELLCT and cell size (CSIZE) to determine the optimum number of cells to provide in order to make effective use of virtual and real storage.

,SCELLCT = *secondary cell count*
specifies the number of cells expected to be in each secondary or non-initial extent of the cell pool. The CPOOL service routine uses SCELLCT and CSIZE to determine the optimum number of cells to provide in order to make effective use of virtual and real storage.

,CSIZE = *cell size*
specifies the number of bytes in each cell of the cell pool. If CSIZE is a multiple of 8, the cell resides on doubleword boundaries. If CSIZE is a multiple of 4, the cell resides on word boundaries. The minimum value of CSIZE is 4 bytes.

,SP = *subpool number*
specifies the subpool from which the cell pool is to be obtained. If a register or variable is specified, the subpool number is taken from bits 24-31.

,LOC=BELOW
,LOC=(BELOW,ANY)
,LOC=ANY
,LOC=(ANY,ANY)
,LOC=RES
,LOC=(RES,ANY)

specifies the location of virtual storage and real storage for the cell pool. This is helpful for users with 24-bit dependencies. The location of real storage specified in this parameter is the location of the storage after it is fixed, either by definition or by PGFIX, PGFIXA, or PGSER. The specification of the LOC parameter, which applies to the location of real storage, is only guaranteed when the area is fixed.

LOC=BELOW indicates that virtual and real storage are to be allocated below 16 megabytes.

LOC=(BELOW,ANY) indicates that virtual storage is to be allocated below 16 megabytes and real storage can be anywhere.

LOC=ANY and LOC=(ANY,ANY) indicate that both virtual and real storage can be located anywhere.

LOC=RES indicates that the location of virtual and real storage depends on the location of the issuer of the macro. If the issuer resides below 16 megabytes, virtual and real storage are allocated below 16 megabytes; if the issuer resides above 16 megabytes, virtual and real storage can be located anywhere.

LOC=(RES,ANY) indicates that the location of virtual storage depends on the location of the issuer of the macro. If the issuer resides below 16 megabytes, virtual storage is allocated below 16 megabytes; if the issuer resides above 16 megabytes, virtual storage is allocated anywhere. Real storage can be located anywhere.

Note: Callers executing in 24-bit addressing mode could perform services for cell pools located in storage above 16 megabytes by specifying LOC=ANY or LOC=(ANY,ANY).

,CPID=*pool id*

specifies the address or register containing the cell pool identifier that is returned to the caller after the pool is created using CPOOL BUILD. The issuer must specify CPID on all subsequent CPOOL requests containing the keywords GET, FREE, or DELETE.

,CELL=*cell addr*

specifies the address or register where the cell address is returned to the user by a GET or a FREE request.

,KEY=*key number*

specifies the key in which storage is to be obtained. If a register is specified, the key is taken from bits 28-31. This parameter is valid for subpools 227, 228, 229, 230, 231, and 241.

,TCB=*tcb addr*

specifies the TCB address for task related storage requests. The TCB must be within the currently addressable address space. If the caller specifies zero as the TCB address, the

CPOOL service routine uses the TCB address in ASCBXTCB. If the CPOOL request is for private area storage and the caller does not specify TCB, the default is the TCB address in PSATOLD.

Note: The TCB resides in storage below 16 megabytes.

,HDR=*hdr*
specifies a 24-byte header, which is placed in the header of each initial and secondary extent. The header can contain user-supplied information that would be useful in a dump.

,LINKAGE=SYSTEM
,LINKAGE=BRANCH
specifies the type of linkage used in CPOOL processing. LINKAGE=SYSTEM indicates that the linkage is via a PC instruction, LINKAGE=BRANCH indicates branch entry. For BUILD and DELETE this processing is between the caller and CPOOL processing; for GET UNCOND, the linkage is within CPOOL processing (that is, between the modules IGVCPOOL and IGVCPEXT).

,REGS=SAVE
,REGS=USE
indicates whether or not registers 2-12 are to be saved. If REGS=SAVE is specified, the registers are saved in a 72-byte user-supplied save area pointed to by register 13. If REGS=USE is specified, the registers are not saved.

Notes:

1. *If GET U,LINKAGE=SYSTEM,REGS=USE is specified, the secondary ASID will not be preserved. In all other cases the secondary ASID is unchanged.*

2. *A program in secondary mode cannot use LINKAGE=SYSTEM.*

The contents of the registers on return from this macro depends on the parameters specified.

Register(s)	Comment
0	Contains the cell pool identification
1	Contains the address of the cell that was obtained if GET unconditional was specified; contains zero if GET conditional was specified and fails
2-12	Saved for BUILD and DELETE requests or if REGS=SAVE is specified
5-13	Saved if GET conditional or FREE is specified with REGS=USE
13	Saved if GET unconditional and REGS=USE is specified or if BUILD or DELETE is specified with either LINKAGE=SYSTEM or LINKAGE=BRANCH

Example 1

Operation: Create a cell pool containing 40-byte cells from subpool 2. Allow for 10 cells in the initial extent and 20 cells in all subsequent extents of the cell pool.

```
CPOOL BUILD,PCELLCT=10,SCELLCT=20,CSIZE=40,SP=2
```

Example 2

Operation: Unconditionally obtain a cell pool, specifying the pool ID in register 2. Use a PC instruction for linkage and do not save the registers.

```
CPOOL GET,U,CPID=(2),REGS=USE,LINKAGE=SYSTEM
```

Example 3

Operation: Free a cell specifying the pool ID in register 2 and the cell address in register 3.

```
CPOOL FREE,CPID=(2),CELL=(3)
```

Example 4

Operation: Delete a cell pool, specifying the pool ID in register 2. Use a PC instruction for linkage.

```
CPOOL DELETE,CPID=(2),LINKAGE=SYSTEM
```

CPOOL (List Form)

The list form of the CPOOL macro instruction builds a non-executable parameter list that can be referred to by the execute form of the CPOOL macro.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede CPOOL.
CPOOL	
b	One or more blanks must follow CPOOL.
BUILD	
,PCELLCT=*primary cell count*	*cell count:* symbol, decimal. **Note:** PCELLCT must be specified on either the list or the execute form of the macro.
,SCELLCT=*secondary cell count*	**Default:** PCELLCT
,CSIZE=*cell size*	*cell size:* symbol, decimal digit. **Note:** CSIZE must be specified on either the list or the execute form of the macro.
,SP=*subpool number*	*subpool number:* symbol, decimal digit. **Default:** SP=0
,LOC=BELOW ,LOC=(BELOW,ANY) ,LOC=ANY ,LOC=RES ,LOC=(RES,ANY)	**Default:** LOC=RES
,CPID=*pool id*	*pool id:* A-type address.
,KEY=*key number*	*key number:* decimal digits 0 - 15.
,TCB=*tcb addr*	*tcb addr:* A-type address or register. **Default:** TCB address in PSATOLD.
,HDR=*hdr*	*hdr:* character string enclosed in single quotes, A-type address.
,MF=L	

The parameters are explained under the standard form of the CPOOL macro instruction with the following exception:

,MF=L
specifies the list form of the CPOOL macro instruction.

CPOOL (Execute Form)

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede CPOOL.
CPOOL	
b	One or more blanks must follow CPOOL.
BUILD	
,PCELLCT=*primary cell count*	*cell count:* symbol, decimal digit, or register (0), (2) - (12). **Note:** PCELLCT must be specified on either the list or the execute format of the macro.
,SCELLCT=*secondary cell count*	**Default:** PCELLCT
,CSIZE=*cell size*	*cell size:* symbol, decimal digit, or register (0), (2) - (12). **Note:** CSIZE must be specified on either the list or the execute form of the macro.
,SP=*subpool number*	*subpool number:* symbol, decimal digit, or register (0), (2) - (12). **Default:** SP=0
,LOC=BELOW ,LOC=(BELOW,ANY) ,LOC=ANY ,LOC=RES ,LOC=(RES,ANY)	**Default:** LOC=RES
,CPID=*pool id*	*pool id:* RX-type address or register (0), (2) - (12).
,KEY=*key number*	*key number:* decimal digits 0 - 15 or register (0), (2) - (12).
,TCB=*tcb addr*	*tcb addr:* RX-type address or register (0), (2) - (12). **Default:** TCB address in PSATOLD.
,HDR=*hdr*	*hdr:* character string enclosed in single quotes, RX-type address, or register (0), (2) - (12).
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	**Default:** LINKAGE=SYSTEM
,MF=*(E, ctrl prog)*	*ctrl prog:* RX-type address or register (0) - (12).

The parameters are explained under the standard form of the CPOOL macro instruction with the following exception:

,MF=(E,*ctrl prog*)
specifies the execute form of the CPOOL macro instruction.

DATOFF - DAT-OFF Linkage

The DATOFF macro transfers control to a specified routine in the DAT-OFF section of the nucleus.

The macro is restricted to key 0, supervisor state users, that are enabled for DAT. Users must include the IHAPSA mapping macro with the DATOFF macro instruction. The macro destroys the contents of general registers 0, 14, and 15.

The DATOFF macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede DATOFF.
DATOFF	
b	One or more blanks must follow DATOFF.
index	**Note:** See the description of the parameters for the valid options.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

index
specifies the routine that is to be given control in the DAT-OFF section of the nucleus. The possible values for *index* along with the entry point in the routine and the purpose of the routine follow.

Index	Entry Point	Purpose
INDCDS	IEAVCDS	DAT-OFF Compare Double and Swap routine
INDMVCL0	IEAVMVC0	General DAT-OFF move character long function
INDMVCLK	IEAVMVKY	General DAT-OFF move character long in user key function
INDXC0	IEAVXC0	General DAT-OFF exclusive OR character function
INDUSR1	IEAVEUR1	User written function
INDUSR2	IEAVEUR2	User written function
INDUSR3	IEAVEUR3	User written function
INDUSR4	IEAVEUR4	User written function

Note: See *SPL: System Modifications* for information about how to insert a user-written function in the nucleus.

,RELATED = *value*

specifies information used to document the macro instruction and to relate the service performed to some corresponding service or function. The format of the information specified can be any valid coding values that the user chooses.

Example 1

Operation: Invoke the general DAT-OFF move character long function. The user must supply the following information in the registers specified:

Registers	Information
2	Location into which the characters are to be moved
3	Length of the area into which the characters are to be moved
4	Location of the area from which the characters are to be moved
5	Length of the area from which the characters are to be moved

Note: Registers 2 and 4 contain real addresses.

```
DATOFF INDMVCL0
```

Example 2

Operation: Invoke the general DAT-Off exclusive OR character function. The user must supply the following information in the registers specified:

Registers	Information
2	Location of the results of exclusive OR character processing
3	Bits 24-31 contain one less than the number of bytes on which the exclusive OR is to be performed.
4	Location of the operand on which the exclusive OR is to be performed

Note: Registers 2 and 4 contain real addresses.

```
DATOFF INDXC0
```

Example 3

Operation: Invoke the general DAT-OFF move character long in user key function. The user must supply the following information in the registers specified:

Registers	Information
2	Location into which the characters are to be moved
3	Length of the area into which the characters are to be moved
4	Location of the area from which the characters are to be moved
5	Length of the area from which the characters are to be moved
6	Bits 24-27 contain the PSW key in which the MVCL is to be executed.

Note: Registers 2 and 4 contain real addresses.

```
DATOFF INDMVCLK
```

DEQ - Release a Serially Reusable Resource

DEQ removes control of one or more serially reusable resources from the active task. Register 15 is set to 0 if the request is satisfied. An unconditional request to release a resource from a task that is not in control of the resource or a request that contains invalid parameters results in abnormal termination of the task.

Note: When global resource serialization is active, the SYSTEM inclusion resource name list and the SYSTEMS exclusion resource name list are searched for every resource specified with a scope of SYSTEM or SYSTEMS. A resource whose name appears on one of these resource name lists might have its scope changed from the scope that appears on the macro instruction. (See *Planning: Global Resource Serialization* for additional information about global resource serialization.)

The description of the entire DEQ macro instruction follows. The DEQ macro instruction also appears in *Supervisor Services and Macro Instructions* with the exception of the RMC, GENERIC, TCB, and UCB parameters. These parameters are restricted in use to programs that run in supervisor state, key 0-7, or with APF authorization, and are, therefore, described only here.

Except for the TCB and UCB, all input parameters to this macro instruction can reside in storage above 16 megabytes for callers executing in 31-bit addressing mode.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede DEQ.
DEQ	
b	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
)	
,RET=HAVE ,RET=NONE	
,RMC=NONE ,RMC=STEP	**Default:** RMC=NONE
,GENERIC=NO ,GENERIC=YES	**Defalult:** 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 are explained as follows.

(
: specifies the beginning of the resource description.

qname addr
: specifies the address in virtual storage of an 8-character name. The *qname* must be the same name specified for the resource in an ENQ macro instruction.

,rname addr
: specifies the address in virtual storage of the name used in conjunction with qname and scope to represent the resource acquired by a previous ENQ macro instruction. The name can be qualified and must be from 1 to 255 bytes long. The rname must be the same name specified for the resource in an ENQ macro instruction.

,

,*rname length*

specifies the length of the *rname* described above. The length must have the same value as specified in the previous ENQ macro instruction. If this parameter is omitted, the assembled length of the *rname* is used. You can specify a value between 1 and 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.

,

,STEP
,SYSTEM
,SYSTEMS

specifies the scope of the resource. You must specify the same STEP, SYSTEM, or SYSTEMS option as you used in the ENQ macro instruction requesting the resource.

)

specifies the end of the resource description.

Note: Multiple resources can be specified with the DEQ macro instruction. You can repeat *qname addr*, *rname addr*, *rname length*, and the scope until there is a maximum of 255 characters including the parentheses.

,RET=HAVE
,RET=NONE

HAVE specifies that the request for releasing the resources named in DEQ is to be honored only if the active task has been assigned control of the resources or if ENQ was executed with ECB. A return code is set if the resource is not held. NONE specifies an unconditional request to release all the resources. RET=NONE is the default. The active task is abnormally terminated if it has not been assigned control of the resources.

In either case, if the resources requested for release were originally queued with the ECB parameter specified, they are released with return code 0.

,RMC=NONE
,RMC=STEP
,GENERIC=NO
,GENERIC=YES

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 resources with the specified *qname* are to be released. In order for the resource to be released, the task must have control of or be in ECB wait for the resource. (ECB was specified on the original ENQ.) If the task is waiting for a resource, but is not in an ECB wait, the task remains queued and waiting.

The following return codes are associated with a GENERIC DEQ:

Hexadecimal Code	Meaning
0	One or more resources which the task had control of or was in ECB wait for have been released.
4	One or more resources were unconditionally requested by the task, but the task was not assigned control. The task is not removed from the wait condition. However, other resources with the same *qname* might have been released.
8	No resources were found for the specified *qname*.

,TCB = *tcb addr*
specifies a register that points to a TCB or specifies the address of a fullword on a fullword boundary that points to a TCB on whose behalf the DEQ is to be done. The caller (not the directed task) is 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, except when ECB was specified on the original ENQ. If ECB was specified on the ENQ and the resource is not owned by the directed task, the TCB DEQ request releases the resources with a zero return code.

Note: The TCB resides in storage below 16 megabytes.

,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. This parameter is used to release a device reserved with the RESERVE macro instruction. The UCB parameter is optional.

Note: The UCB resides in storage below 16 megabytes.

,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 can be any valid coding values.

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 4.

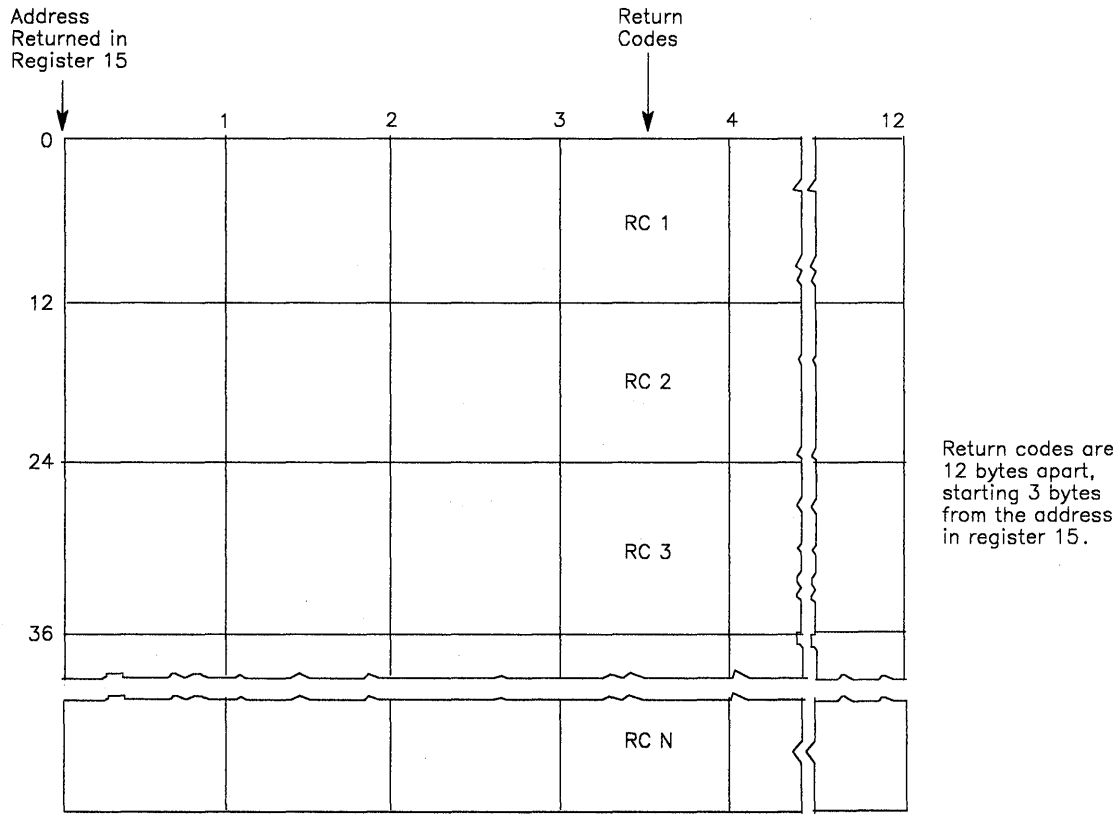


Figure 4. Return Code Area Used by DEQ

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)
```

Example 4

Operation: Release control of the resource in Example 1 of ENQ, if it has been assigned to the current TCB. The length of the rname is explicitly defined as 8 characters.

```
DEQ (MAJOR1,MINOR1,8,STEP),RET=HAVE
```

DEQ (List Form)

Use the list form of the DEQ macro instruction to construct a control program parameter list. The number of *qname, rname*, and scope combinations in the list form of DEQ must be equal to the maximum number of *qname, rname*, and scope combinations in any execute form of DEQ that refers to that list form. The list form of the DEQ macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede DEQ.
DEQ	
b	One ore 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
)	
,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=0	**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 are explained under the standard form of the DEQ macro instruction, with the following exception:

,MF=L
specifies the list form of the DEQ macro instruction.

DEQ (Execute Form)

A remote control program parameter list is used in, and can be modified by, the execute form of the DEQ macro. The parameter list can be generated by the list form of either the DEQ or the ENQ macro instruction.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede DEQ.
DEQ	
b	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 digits, or register (2) - (12).
, ,STEP ,SYSTEM ,SYSTEMS	
)	**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) - (12).

The parameters are explained under the standard form of the DEQ macro instruction, with the following exception:

,MF = (E,*ctrl addr*)
specifies the execute form of the DEQ macro instruction using a remote control program parameter list.

DOM - Delete Operator Message

The DOM macro instruction is used to delete an operator message or group of messages from the display screen of the operator's console. It can also prevent messages from ever appearing on any operator's console. When a program no longer requires that a message be displayed, it can issue the DOM macro instruction to delete the message.

Depending on the timing of the DOM relative to the WTO(R), the message may or may not be displayed. If the message is being displayed, it is removed when space is required for other messages. If the message is not yet displayed, it is removed before it gets displayed.

When a WTO or WTOR macro instruction is issued, the system assigns an identification number to the message and returns this number (32 bits right-justified) to the issuing program in register 1. When the display of this message is no longer needed, the issuing program can issue the DOM macro instruction using the identification number that was returned in general register 1.

The DOM macro instruction is written as follows:

```
    name                    name: symbol. Begin name in column 1.
b                           One or more blanks must precede DOM.
DOM
b                           One or more blanks must follow DOM.

MSG=addr                    addr: register (1) - (12), or an address.
MSGLIST=list addr           list addr: symbol, RX-type address, or register (1) - (12).
TOKEN=addr                  addr: register (1) - (12), or an address.
DOMCBLK=addr                addr: register (1) - (12), or an address
    ,COUNT=addr             addr: register (2) - (12), or an address.
    ,SYSID=addr             addr: register (2) - (12), or an address.
    ,REPLY=YES
    ,SCOPE=SYSTEM
    ,SCOPE=SYSTEMS
```

The parameters are explained as follows:

MSG=

The field or register contains the message id of a message to be deleted.

MSGLIST=

specifies the address of a list of one or more fullwords, each word containing the message id of a message to be deleted.

REPLY=

specifies that one or more WTOR messages are to be deleted. REPLY is not required, and is invalid with DOMCBLK, TOKEN, SYSID, COUNT and SCOPE.

DOMCBLK=

specifies the address of a DOM control block that is to be used as input for the DOM macro. Only authorized programs can issue DOMCBLK, which is mutually exclusive with all keywords except for SCOPE.

TOKEN=

specifies a field or register containing a 4-byte token that is associated with messages to be deleted. When you issue WTO or WTOR to write a message, you can choose a token value, and specify it as an input parameter to WTO(R) via the TOKEN keyword. WTO(R) returns control to the application with a message id in register 1. To delete the message by the TOKEN method, ignore the message id returned by WTO(R) in register 1, and specify the token value instead, using the TOKEN keyword when you issue DOM. TOKEN is an alternate method for identifying messages, which is independent of the register 1 message id.

Authorized users may delete any messages originally issued under the same ASID and system id with this keyword. Unauthorized users may delete only those messages that were originally issued under the same jobstep TCB, ASID, and system id. The value of the token may not be the same as the id that was returned in register 1 after a WTO or WTOR. TOKEN is mutually exclusive with MSG, MSGLIST, COUNT, DOMCBLK, and REPLY.

SYSID=

specifies a field or register containing the 1-byte id of the system on which the message was issued. If no message ids are specified, (that is, MSG or MSGLIST is not specified) all messages issued from the specified system are deleted. If message ids are specified, (i.e., MSG or MSGLIST has been specified), messages indicated by the MSG or MSGLIST keyword issued from the specified system are deleted.

SYSID is invaliid with DOMCBLK, COUNT, and REPLY. SYSID can be used with the TOKEN keyword to delete all messages originally issued from a particular system with the specified TOKEN. Authorized users may delete any messages originally issued under the same ASID when TOKEN and SYSID are specified. Unauthorized users may delete only those messages that were originally issued under the same jobstep TCB and ASID when TOKEN and SYSID are specified. If an address is used, the address points to a 1-byte field which contains the system id.

COUNT=

specifies a field or register containing the one-byte count of 4-byte message ids associated with this request. The count must be from 1 to 60. If COUNT is specified, the issuer must not set the high order bit on in the last entry of the DOM parameter list (DOMPL). If COUNT is not specified, the message ids are treated as 3-byte ids. If an address is used, the address points to a 1-byte field that contains the count. COUNT is invalid with DOMCBLK, SYSID, TOKEN, and REPLY.

SCOPE=SYSTEM
SCOPE=SYSTEMS
specifies how to process the DOM request. If SCOPE=SYSTEMS is specified, the DOM request is to be communicated to other processors. If SCOPE=SYSTEM is specified, the DOM request is not to be communicated to other processors. If SCOPE is not specified, the DOM request defaults to SCOPE=SYSTEMS.

Notes:

1. *For any DOM keywords that allow a register specification, the value must be right-justified in the register and the remaining bytes within the register must be zero.*

2. *Any authorized DOM keywords that are specified by an unauthorized program will cause a 157 ABEND.*

Example 1

Operation: Delete an operator message. The message id is in register 1.

```
DOM MSG=(1)
```

Example 2

Operation: Delete a list of operator messages.

```
DOM MSGLIST=ID2
```

Example 3

Operation: Delete four operator messages. The number of messages to be deleted is stored in the field named FOUR, and ID3 is the address of the list of message ids for the four messages.

```
DOM MSGLIST=ID3,COUNT=FOUR
```

Example 4

Operation: Delete a single message issued on a particular system. The message ID is in register 1, and the one-byte system id is stored in the field named TWO.

```
DOM MSG=(1),SYSID=TWO
```

Example 5

Operation: Delete all messages issued on a particular system. The one-byte system id is stored in the field named SYSNAME.

```
DOM SYSID=SYSNAME
```

Example 6

Operation: Delete all messages issued with a particular token on a particular system. The four-byte token is stored in TOKEN1, and the one-byte system id is in TWO.

```
DOM TOKEN=TOKEN1,SYSID=TWO
```

DSGNL - Issue Direct Signal

The DSGNL macro instruction uses the signal processor (SIGP) instruction to modify or sense the physical state of a specific processor in a multiprocessing configuration. The SIGP instruction order codes specified on the DSGNL macro instruction are defined as direct services. Additional SIGP order codes defined as remote services are available through the RISGNL and RPSGNL macro instructions. See *Principles of Operations* for an explanation of the order codes.

Programs executing in cross memory mode can issue this macro instruction.

The DSGNL macro instruction is written as follows:

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

SENSE START STOP RESTART SSS ICPUR CPUR STATUS PREFIX (0)	
,CPU = *PCCA addr*	*PCCA addr:* RX-type address, or register (1).
,PARAM = *addr* ,PARAM = (2)	*addr:* RX-type address, or register (2). **Note:** This parameter is required with PREFIX and STATUS only. It cannot be specified with any of the other parameters.

The parameters are explained as follows:

SENSE
START
STOP
RESTART
SSS
ICPUR
CPUR
PREFIX
STATUS
(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. (Only the direct class functions are valid.) The actions and codes are:

	Order Code	Action
SENSE	01	State of specified processor is to be sensed
START	04	Start function
STOP	05	Stop function
RESTART	06	Restart function
SSS	09	Stop and store status function
ICPUR	0B	Initial processor reset function
CPUR	0C	Processor reset function
PREFIX	0D	Set prefix from address
STATUS	0E	Store status at address

,CPU=PCCA addr

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

Note: The PCCA resides in storage below 16 megabytes.

,PARAM=*addr*
,PARAM=(2)

allows an address to be passed to the specified processor. If *addr* is coded, the word at that location is loaded into register 2 and passed to the specified processor. The contents of that location must contain a real address. If (2) is coded, the contents of register 2 is passed to the processor. Register 2 must also contain a real address.

When this parameter is used with PREFIX, the word passed to the specified processor is the address to which the processor's prefix register is to be set.

When this parameter is used with STATUS, the word passed to the specified processor is the real address at which the processor's status is to be stored.

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 the 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 order code.
08	Function unsuccessfully initiated or successful SIGP SENSE request. Status is returned in register 0.
0C	Specified processor is either not installed, not configured into the system, or powered off.
14	MSSF is currently inoperative.

With a return code of 8, register 0 contains status information from the SIGP macro instruction. The bit settings and meanings follow:

Bits	Meaning
0	Equipment check
1-21	Unassigned, contains zeros
22	Incorrect state
23	Invalid parameter
24	External call pending
25	Stopped
26	Operator intervening
27	Check stop
28	Not ready
29	MSSF currently inoperative
30	Invalid order code
31	Receiver check

Example 1

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

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

DYNALLOC - Dynamic Allocation

See Volume 1 for the description of this macro instruction.

ENQ - Request Control of a Serially Reusable Resource

ENQ requests the control program to assign control of one or more serially reusable resources to a task. If any of the resources are not available, the task might be placed in a wait condition until all of the requested resources are available. Once control of a resource has been assigned to a task, it remains with that task until one of the programs of the same task issues a DEQ macro instruction specifying the same resource. Register 15 is set to 0 if the request is satisfied.

You can also use ENQ to determine the status of the resource; whether it is immediately available or in use, and whether control of the resource has been previously requested by the active task in another ENQ macro instruction.

You can request either shared or exclusive use of a resource. The resource is represented in the ENQ by a pair of names, the *qname* and the *rname*, and a scope value. The scope value determines the scope of serialization; that is, what other tasks, address spaces, or systems can use the resource. The control program does not correlate the names with the actual resources. ENQ simply coordinates access to whatever it is the names represent. The names may be given meaning within a job step or across job steps. In either case, all programs for which coordination of the resource is provided must refer to it by the same name and scope value. You must ensure that the name and scope value are used consistently.

Issuing two ENQ macro instructions for the same resource without an intervening DEQ macro instruction results in abnormal termination of the task, unless the second ENQ designates RET=TEST, USE, CHNG, or HAVE. If normal termination of a task is attempted while the task still has control of any serially reusable resources, all requests made by this task will be automatically dequeued. If resource input addresses are incorrect, the task is abnormally terminated.

Global resource serialization counts and limits the number of concurrent resource requests in an address space. If an unconditional ENQ (an ENQ that uses the RET=NONE option) causes the count of global resource serialization requests to exceed the sum of a threshold value plus a tolerance value, an authorized caller is abended with a system code of X'538'. See "Limiting Global Resource Serialization Requests" in Volume 1.

Note: When global resource serialization is active, the SYSTEM inclusion resource name list and the SYSTEMS exclusion resource name list are searched for every resource specified with a scope of SYSTEM or SYSTEMS. A resource whose name appears on one of these resource name lists might have its scope changed from the scope that appears on the macro instruction. (Refer to *Planning: Global Resource Serialization* for additional information.)

Except for the TCB, all input parameters to this macro instruction can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

An ENQ used with the MASID and MTCB operands provides a special form of the ENQ macro instruction that allows a further conditional control of a resource. One task, called the

"issuing task" can issue an ENQ macro for a resource specifying the ASID and TCB of another task, called the "matching task." The MTCB and MASID operands are specified with RET=HAVE, RET=TEST, and/or ECB= to provide additional return codes. If the issuing task does not acquire control of the resource, it may receive a return code indicating that the resource is controlled by the matching task. Upon receiving this return code, the issuing task could use the resource, if serialization between itself and the matching task has been accomplished by some pre-arranged protocol known to both the issuing and matching tasks.

The description of the ENQ macro instruction follows. The ENQ macro instruction is also described in *Supervisor Services and Macro Instructions* with the exception of the SMC, ECB, and TCB parameters. These parameters are restricted in use to programs that run in supervisor state, PSW key 0-7, or APF authorized and are therefore only described here.

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

(	
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
)	
,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. ECB and TCB can be specified together. If TCB is specified but not ECB, then RET=CHNG, TEST or USE must be specified above.
,MASID=*matching-asid addr* ,MTCB=*matching-tcb addr*	*matching-asid addr:* A-type address, or register (2)-(12). *matching-tcb addr:* A-type address, or register (2)-(12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

(
: specifies the beginning of the resource description.

qname addr
: specifies the address in virtual storage of an 8-character name. Every program issuing a request for a serially reusable resource must use the same *qname, rname*, and scope 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. If the name specified as *rname* is defined by an EQU assembler instruction, *rname length* must be specified.

,
,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 and 255 to override the assembled length. If the name specified as *rname*, is defined by an EQU assembler instruction, *rname length* must be specified.

,
,STEP
,SYSTEM
,SYSTEMS
: specifies the scope of the resource.

  STEP specifies that the resource can be used only within an address space. If STEP is specified, a request for the same *qname* and *rname* from a program in another address space denotes a different resource.

  SYSTEM specifies that the resource can be used by more than one address space.

  SYSTEMS specifies that the resource can be shared between systems.

  STEP, SYSTEM, and SYSTEMS are mutually exclusive and do not refer to the same resource. If two macro instructions specify the same *qname* and *rname*, but one specifies STEP and the other specifies SYSTEM or SYSTEMS, they are treated as requests for different resources.

  When global resource serialization is active, scope conversion can occur. This could result in two requests with different scopes referring to the same resource. See *Planning: Global Resource Serialization* for details.

specifies the end of the resource description.

Note: Multiple resources can be specified in the ENQ macro instruction. You can repeat the *qname addr*, *rname addr*, type of control, *rname length*, and scope until there is a maximum of 255 characters including the parentheses.

,RET=CHNG
,RET=HAVE
,RET=TEST
,RET=USE
,RET=NONE
specifies the type of request for all of the resources named above.

CHNG - the status of the resource specified is changed from shared to exclusive control.

HAVE - control of the resources is requested conditionally; that is, control is requested only if a request has not been made previously for the same task.

TEST - the availability of the resources is to be tested, but control of the resources is not requested.

USE - control of the resources is to be assigned to the active task only if the resources are immediately available. If any of the resources are not available, the active task is not placed in a wait condition.

NONE - control of all the resources is unconditionally requested.

,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).

When SMC=STEP is specified with RET=HAVE and the requesting task already has control of the resource, the SMC function is turned on and the task continues to control the resource.

SMC= and TCB= are mutually exclusive with the MASID parameter, therefore, hexadecimal return codes 20, 24, 28, and 44 will not be given by an ENQ using the SMC or TCB operands.

The return codes and status of the set must-complete function for the various RET= specifications are as follows:

	Hexadecimal Code	SMC Status
RET=CHNG	0	on
	4	off
	8	off
	14	off
RET=HAVE	0	on
	8	on
	14	off
RET=TEST	0	off
	4	off
	8	off
	14	off
RET=USE	0	on
	4	off
	8	off
	14	off
	18	off

ECB specifies the address of an ECB, and conditionally requests all of the resources named in the macro instruction. If the return code for one or more requested resources is hexadecimal 4 or 24 and the request is not nullified by a corresponding DEQ, the ECB is posted when all the requested resources (specifically, those that initially received a return code of 4 or 24) are assigned to the requesting task.

If the ECB parameter is an A-type address, the address is the name of the fullword that is used as an ECB. If the operand is a register, then the register contains the address of the ECB.

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

Note: The TCB resides in storage below 16 megabytes.

,MASID=*matching-asid addr*
specifies the matching task (by defining a matching ASID) for the ENQ, if used in conjunction with the MTCB parameter. MASID defines the ASID of a task that may be using a resource desired by the issuer of the ENQ macro instruction. If the MASID parameter is an A-type address, the address is the name of a fullword containing the ASID. If the operand is a register, then the register contains the ASID.

Note: MASID can only be specified if MTCB is also specified.

,MTCB=*matching-tcb addr*
specifies the matching task (by defining a matching TCB) for the ENQ, if used in conjunction with the MASID parameter. MTCB defines the TCB of a task that may be using a resource desired by the issuer of the ENQ macro instruction.

If the task specified by the MASID and MTCB parameters is not using the resource, global resource serialization gives control to the issuer of the ENQ and returns a return code indicating whether the resource can be used. If the task specified by MASID and MTCB parameters is using the resource, global resource serialization records a request for the resource, suspends the issuing task until the resource is available, or optionally returns a return code indicating that an ECB will be posted when the resource can be used.

The MASID and MTCB parameters are specified with RET=HAVE, RET=TEST, and/or ECB= parameters to elicit additional return codes that provide information about the owner of the resource. If the MTCB parameter is an A-type address, the address is the name of a fullword containing the TCB. If the operand is a register, then the register contains the TCB.

Note: MTCB can only be specified if MASID is also specified.

,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.

Return codes are provided by the control program only if you specify RET=TEST, RET=USE, RET=CHNG, 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 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 5.

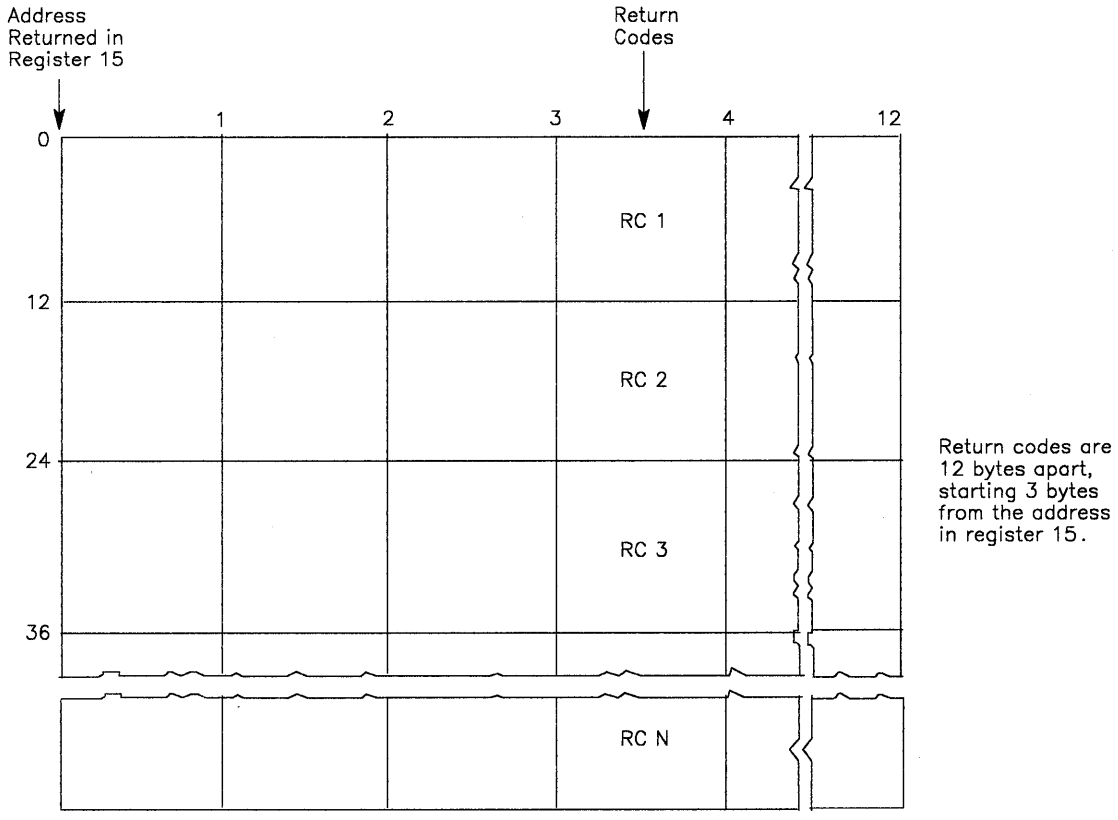


Figure 5. Return Code Area Used by ENQ

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 is 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. The ECB is not posted.
4	For RET=TEST or RET=USE, the resource is not immediately available. For RET=CHNG, the status cannot be changed to exclusive. 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. The task has control of resource. For RET=CHNG, the resource has not been enqueued. If bit 3 is on -- shared control of resource; if bit 3 of the first byte of the ENQ parameter list is off -- exclusive control. The ECB is not posted.
14	A previous request for control of the same resource has been made for the same task. The task does not have control of resource. The ECB is not posted.
18	For RET=HAVE, RET=USE, or ECB=, the limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource. The ECB is not posted.
20	The matching task (the task specified in the MASID/MTCB parameters) owns the resource. The issuer of the ENQ macro instruction may use the resource but it must ensure that the owning task does not terminate while the issuer of the ENQ macro is using the resource. If the issuer of the ENQ requested exclusive control, then this return code indicates that the matching task is the only task that currently owns the resource. If the issuer of the ENQ requested shared control and the owning task had requested shared control, this return code may indicate that a previous task had requested exclusive control. The issuing task must issue a DEQ to cancel this ENQ. The ECB will not be posted.
24	The issuing task will have exclusive control after the ECB is posted. The issuing task may use the resource but must ensure that the matching task does not terminate while the issuing task is using the resource. The issuing task must issue a DEQ to cancel the ENQ.
28	The issuing task cannot obtain exclusive control of the resource using the MASID/MTCB ENQ. The matching task's involvement with other tasks precludes control by the issuing task. This task must not issue a DEQ to cancel the ENQ. The ECB will not be posted.
44	The issuing task is violating a restriction of the MASID/MTCB ENQ in one or more of the following ways: • Another task has already issued this ENQ for this resource specifying the same MASID/MTCB. • The MASID/MTCB parameters specify a task that acquired control of the resource by using the MASID/MTCB ENQ. • The matching task requested ownership of the resource but has not yet been granted ownership. The ECB will not be posted. Return code 44 is never given by an ENQ RET=TEST, return code 4 is given instead.

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 sharable 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)

Use the list form of ENQ to construct a control program parameter list. Any number of resources can be specified in the ENQ macro instruction, therefore, the number of *qname, rname*, and scope combinations in the list form of the ENQ macro instruction must be equal to the maximum number of *qname, rname*, and scope combinations in any execute form of the macro instruction that refers to that list form.

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

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

(	
qname addr	*qname addr:* A-type address.
,	
,*rname addr*	*rname addr:* A-type address.
,	**Default:** E
,E	
,S	
,	
,*rname length*	*rname length:* symbol or decimal digit.
	Default: assembled length of *rname*
,	**Default:** STEP
,STEP	
,SYSTEM	
,SYSTEMS	
)	
,RET=CHNG	**Default:** RET=NONE
,RET=HAVE	
,RET=TEST	
,RET=USE	
,RET=NONE	
,SMC=NONE	
,SMC=STEP	
,ECB=*ecb addr*	*ecb addr:* A-type address.
,TCB=0	**Default:** SMC=NONE
	Note: ECB cannot be specified with RET above.
	Note: TCB or ECB must be specified on the list form if it is used on the execute form. ECB and TCB can be specified together. If TCB is specified but not ECB, then RET=CHNG, TEST or USE must be specified above.
,MASID=*0*	
,MTCB=*0*	
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

The parameters are explained under the standard form of the ENQ macro instruction, with the following exception:

,MF = L
specifies the list form of the ENQ macro instruction.

The list form of this macro generates a prefix followed by the parameter list, however the label specified in MF = L does not include an offset prefix area. If MASID, MTCB, TCB, or ECB is specified, these labels are offset; allowance must be made for the parameter list prefix.

ENQ (Execute Form)

A remote control program parameter list is used in and can be modified by the execute form of the ENQ macro instruction. The parameter list can be generated by the list form of ENQ.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ENQ.
ENQ	
b	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, the (,), 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	
)	**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:* RX-type address, or register (2) - (12). *tcb addr:* RX-type address, or register (2) - (12). **Note:** ECB cannot be specified with RET above. **Note:** ECB and TCB can be specified together. If TCB is specified but not ECB, then RET=CHNG, TEST or USE must be specified above.
,MASID=*matching-asid addr* ,MTCB=*matching-tcb addr*	*matching-asid addr:* Rx-type address, or register (2)-(12). *matching-tcb 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) - (12).

The parameters are explained under the standard form of the ENQ macro instruction, with the following exceptions:

,MF=(E,*ctrl addr)*

specifies the execute form of the ENQ macro instruction using a remote control program parameter list.

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

The list form of this macro generates a prefix followed by the parameter list, however the label specified in MF=L does not include an offset prefix area. If MASID, MTCB, TCB, or ECB is specified, these labels are offset; allowance must be made for the parameter list prefix.

ESPIE - Extended SPIE

The ESPIE macro instruction extends the function of the SPIE (specify program interruption exits) macro instruction to callers in 31-bit addressing mode. Callers in either 24-bit or 31-bit addressing mode can issue the ESPIE macro instruction. Only callers in 24-bit addressing mode can issue the SPIE macro instruction. For additional information concerning the relationship between the SPIE and the ESPIE macro instructions, see the section "Interruption Services" in Volume 1.

The ESPIE macro instruction performs the following functions using the options specified:

- Establishes an ESPIE environment (that is, identifies the interruption types that are to cause entry to the ESPIE exit routine) by executing the SET option of the ESPIE macro instruction.
- Deletes an ESPIE environment (that is, cancels the current SPIE/ESPIE environment) by executing the RESET option of the ESPIE macro instruction
- Determines the current SPIE/ESPIE environment by executing the TEST option of the ESPIE macro instruction

The following description of the ESPIE macro instruction also appears in *Supervisor Services and Macro Instructions* with the exception of interruption type 17. This interruption type designates page faults and its use is restricted to an installation-authorized system programmer.

SET Option

The SET option of the ESPIE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ESPIE.
ESPIE	
b	One or more blanks must follow ESPIE.
SET	
,exit addr	*exit addr:* A-type address or register (2) - (12).
,(interruptions)	*interruptions:* decimal numbers 1 - 15 or 17 expressed as single values: (2, 3, 4, 7, 8, 9, 10) ranges of values: ((2, 4), (7, 10)) combinations: (2, 3, 4, (7, 10))
,PARAM=*list addr*	*list addr:* A-type address or register (2) - (12).

The parameters are explained as follows:

SET
indicates that an ESPIE environment is to be established.

,exit addr
specifies the address of the exit routine to be given control when program interruptions of the type specified by *interruptions* occur. The exit routine will receive control in the same addressing mode as the issuer of the ESPIE macro instruction.

,(interruptions)
indicates the interruption types that are being trapped. 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

These 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.

If a program interruption type is maskable, the corresponding program mask bit in the PSW is set to 1. If a maskable interruption is not specified, the corresponding bit in the PSW is set to 0. Interruption types not specified above (except for type 17) are handled by the control program. The control program forces an abend with the program check as the completion code. If an ESTAE-type recovery routine is also active, the SDWA indicates a system-forced abnormal termination. The registers at the time of the error are those of the control program.

Note: For both ESPIE and SPIE – If you are using vector instructions and an interruption of 8, 12, 13, 14, or 15 occurs, your recovery routine can check the exception extension code (the first byte of the two-byte interruption code in the EPIE or PIE) to determine whether the exception was a vector or scalar type of exception.

,PARAM=*list addr*
specifies the fullword address of a parameter list that is to be passed by the caller to the exit routine.

On return from the SET option of the ESPIE macro instruction, the registers contain the following information:

Register	Content
0	Unpredictable
1	Token representing the previously active SPIE/ESPIE environment
2-13	Unchanged
14	Unpredictable
15	Return code of 0

Example 1

Operation: Give control to an exit routine for interruption types 1 and 4. EXIT is the location of the exit routine to be given control and PARMLIST is the location of the user-parameter list to be used by the exit routine.

```
ESPIE SET,EXIT,(1,4),PARAM=PARMLIST
```

Example 2

Operation: Give control to the exit routine located at EXIT when a page fault occurs.

```
ESPIE SET,EXIT,(17)
```

RESET Option

The RESET option of the ESPIE routine cancels the active SPIE/ESPIE environment and restores the SPIE/ESPIE environment specified by *token*.

The RESET option of the ESPIE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ESPIE.
ESPIE	
b	One or more blanks must follow ESPIE.
RESET	
,token	*token:* RX-type address or register (1) or (2) - (12).

The parameters are explained as follows:

RESET
indicates that the current ESPIE environment is to be deleted and the previously active SPIE/ESPIE environment specified by *token* is to be re-established.

,*token*
specifies a fullword that contains a token representing the previously active SPIE/ESPIE environment. This is the same token that ESPIE processing returned to the caller when the ESPIE trap was established using the SET option of the ESPIE macro instruction.

If the token is zero, all SPIEs and ESPIEs are deleted.

On return from ESPIE RESET, the contents of the registers are as follows:

Register	Contents
0	Unpredictable
1	Token identifying the new active SPIE/ESPIE environment
2-13	Unchanged
14	Unpredictable
15	Return code of 0

Example 1

Operation: Cancel the current SPIE/ESPIE environment and restore the SPIE/ESPIE environment represented by the contents of TOKEN.

```
ESPIE RESET,TOKEN
```

TEST Option

The TEST option of the ESPIE macro instruction determines the active SPIE/ESPIE environment and returns the information in a four-byte parameter list.

The TEST option of the ESPIE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ESPIE.
ESPIE	
b	One or more blanks must follow ESPIE.
TEST	
,parm addr	*parm addr:* RX-type address, or register (1) or (2) - (12).

The parameters are explained as follows:

TEST
indicates a request for information concerning the active or current SPIE/ESPIE environment. ESPIE processing returns this information to the caller in a four-word parameter list located at *parm addr*.

,parm addr
specifies the address of a four-word parameter list aligned on a fullword boundary. The parameter list has the following form:

Word	Content
0	Address of the user-exit routine (31-bit address with the high-order bit set to 0)
1	Address of the user-defined parameter list
2	Mask of program interruption types
3	Zero

On return from ESPIE TEST, the registers contain the following information:

Register	Contents
0	Unpredictable
1-13	Unchanged
14	Unpredictable
15	Return code as follows:

Code	Meaning
0	An ESPIE exit is active and the four-word parameter list contains the the information specified in the description of the *parm addr* parameter.
4	A SPIE exit is active. Word 1 of the parameter list described under *parm addr* contains the address of the current PICA. Words 0, 2, and 3 of the parameter list are unpredictable.
8	No SPIE or ESPIE is active. The contents of the four-word parameter list are unpredictable.

Example 1

Operation: Identify the active SPIE/ESPIE environment. Return the information about the exit routine in the four-word parameter list, PARMLIST. Also return, in register 15, an indication of whether a SPIE, ESPIE, or neither is active.

```
ESPIE TEST,PARMLIST
```

ESPIE (List Form)

The list form of the ESPIE macro instruction builds a non-executable problem program parameter list that can be referred to or modified by the execute form of the ESPIE macro instruction.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ESPIE.
ESPIE	
b	One or more blanks must follow ESPIE.
SET	
,exit addr	*exit addr:* A-type address. **Note:** This parameter must be specified on either the list or the execute form of the macro instruction.
,(interruptions)	*interruptions:* decimal number 1 - 15 or 17 expressed as single values: (2, 3, 4, 7, 8, 9, 10) ranges of values: ((2, 4), (7, 10)) combinations: (2, 3, 4, (7, 10))
,PARAM=*list addr*	*list addr:* A-type address.
,MF=L	

The parameters are explained under the standard form of the ESPIE macro instruction with the following exception:

,MF=L
specifies the list form of the ESPIE macro instruction.

Example 1

Operation: Build a non-executable problem program parameter list that will cause control to be transferred to the exit routine, EXIT, for the interruption types specified in the execute form of the macro instruction. Provide the address of the user parameter list, PARMLIST.

```
LIST1 ESPIE SET,EXIT,,PARAM=PARMLIST,MF=L
```

ESPIE (Execute Form)

The execute form of the ESPIE macro instruction can refer to and modify the parameter list constructed by the list form of the ESPIE macro instruction.

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

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

SET	
,exit addr	*exit addr:* RX-type address or register (2) - (12). **Note:** This parameter must be specified on either the list or the execute form of the macro instruction.
,(interruptions)	*interruptions:* decimal number 1 - 15 or 17 expressed as single values: (2, 3, 4, 7, 8, 9, 10) ranges of values: ((2, 4), (7, 10)) combinations: (2, 3, 4, (7, 10))
,PARAM=*list addr*	*list addr:* RX-type address or register (1) or (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 ESPIE macro instruction with the following exception:

,MF=(E,*ctrl addr*)
specifies the execute form of the ESPIE macro instruction using a remote control program parameter list.

Example 1

Operation: Give control to a user-exit routine for interruption types 1, 4, 6, 7, and 8. The exit routine address and the address of a user-parameter list for the exit routine are provided in a remote control program parameter list at LIST1.

```
ESPIE SET,,(1,4,(6,8)),MF=(E,LIST1)
```

ESTAE - Specify Task Abnormal Exit Extended

This macro can be assembled compatible between MVS/XA and MVS/370 through the use of the SPLEVEL macro instruction. Default processing will result in an expansion of the macro that operates only with MVS/XA. See the topic "Selecting the Macro Level" for additional information. If you are executing in 31-bit addressing mode, you must use the MVS/XA version of this macro instruction.

The ESTAE macro instruction allows the user to intercept a scheduled ABEND. Control is given to a user-specified recovery routine in which the user can, for example, perform pre-termination processing, diagnose the cause of ABEND, and specify a retry address if he wishes to avoid the termination. These recovery routines operate in both problem program and supervisor modes.

The addressing mode in which the ESTAE macro expansion executes becomes the addressing mode in which the ESTAE exits and retry routines execute (that is, the ESTAE exits and retry routines execute in the same addressing mode as the issuer of the ESTAE macro instruction.)

Note: The ESTAE macro instruction is not supported in cross memory mode.

The description of the ESTAE macro instruction follows. The ESTAE macro instruction is also described in *Supervisor Services and Macro Instructions* with the exception of the BRANCH, SVEAREA, KEY, RECORD, AND TOKEN parameters. These parameters are restricted in use, and, therefore, are described only in here.

"ESTAE-Type Recovery Routines" in Volume 1 describes the complete interface to the ESTAE exit routine.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ESTAE.
ESTAE	
b	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 addr*	**Default:** BRANCH=NO *save addr:* A-type address, or register (2) - (12) or (13).
,KEY=SAVE ,KEY=*storage key*	*storage key:* any numeral in the range 0-15.
,RECORD=NO ,RECORD=YES	**Default:** RECORD=NO
,TOKEN=*token addr*	*token addr:* A-type address, or register (2) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows.

exit addr
0

specifies the 31-bit address of an ESTAE recovery routine to be entered if the task issuing this macro instruction terminates abnormally. The recovery routine executes in the addressing mode of the issuer of the ESTAE. If 0 is specified, the most recent ESTAE routine is canceled.

,CT
,OV

specifies the creation of a new ESTAE exit (CT) or indicates that parameters passed in this ESTAE macro instruction are to overlay the data contained in the previous ESTAE routine (OV).

,PARAM=*list addr*

specifies the 31-bit address of a user-defined list containing data to be used by the ESTAE routine when it is scheduled for execution.

,XCTL=NO
,XCTL=YES

specifies that the ESTAE macro instruction will be canceled (NO) or will not be canceled (YES) if an XCTL macro instruction is issued by this program.

,PURGE=NONE
,PURGE=QUIESCE
,PURGE=HALT

specifies that all outstanding requests for I/O operations are not to be saved when the ESTAE routine gets control (HALT) or that I/O processing is to be allowed to continue normally when the ESTAE routine gets control (NONE) or that all outstanding requests for I/O operations are to be saved when the ESTAE routine is taken (QUIESCE). If QUIESCE is specified, the user's retry routine can restore the outstanding I/O requests.

PURGE=NONE specifies that all control blocks affected by input/output processing can continue to change during ESTAE routine processing. If you specify PURGE=NONE, and the ABEND was originally scheduled because of an error in input/output processing, an ABEND recursion develops when an input/output interruption occurs, even if the ESTAE routine is in progress. Thus, it will appear that the ESTAE routine failed when, in reality, input/output processing caused the failure.

Note: If you specify PURGE=HALT while using ISAM:

- 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 might be destroyed if the data set is being updated or added to when the failure occurred.

,ASYNCH=YES
,ASYNCH=NO

specifies that asynchronous exit processing will be allowed (YES) or prohibited (NO) while the user's ESTAE routine 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 ESTAE 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 ESTAE routine issues the CHECK macro instruction 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 exit handling was the cause of the failure.

,TERM=NO
,TERM=YES
specifies that the ESTAE routine will be scheduled (YES) or will not be scheduled (NO) in the following situations:

- Cancel by operator
- Forced logoff
- Expiration of job step timer
- Exceeding of wait time limit for job step
- ABEND condition because of DETACH of an incomplete subtask when the STAE option was not specified on the DETACH
- ABEND of the attaching task when the ESTAE macro instruction was issued by a subtask
- ABEND of job step task when a non-job step task requested ABEND with the STEP option.

When the ESTAE routine is entered because of one of the preceding reasons, re-try is not permitted. If a dump is requested at the time of ABEND, it is taken before entry into the ESTAE routine.

Note: If DETACH was issued with the STAE parameter, the following occurs for the task to be detached:

- All ESTAE routines are entered.
- The most recently established STAE routine is entered.
- All STAI/ESTAI routines are entered unless one of the STAI routines issues return code 16.

In these cases, entry to the routine occurs before dumping and re-try is not permitted.

,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 save area is a 72-byte area used to save the general registers. If the caller is not in key zero, the KEY parameter must be specified.

,KEY=SAVE
,KEY=*storage key*
specifies that supervisor state users who are not in key zero can use the branch entry interface to the ESTAE service routine.

If the user specifies KEY=SAVE, the system saves the current PSW protection key in register 2 and issues a set protection key instruction (SPKA) to change to protection key zero. When the ESTAE service routine returns control, it restores the original PSW key from register 2. Therefore, the user should save register 2 before the macro expansion

and restore it afterwards. Specifying KEY=SAVE destroys the contents of register 2 during the macro expansion.

On the other hand, if the user knows the current PSW protection key, he may specify it directly in the form KEY=(0-15) to eliminate saving and restoring the original protection key. This procedure eliminates an IPK instruction and prevents the use of register 2 in the macro expansion.

,RECORD=NO
,RECORD=YES
specifies that the system diagnostic work area (SDWA) is not to be written to SYS1.LOGREC (NO) or that the entire SDWA (including the fixed length base, the variable length recording area, and the recordable extensions) is to be written to SYS1.LOGREC (YES).

,TOKEN=*token addr*
specifies that a four-byte token is to be associated with the ESTAE routine. Unauthorized or accidental destruction of the ESTAE routine is prevented because the ESTAE cannot be canceled or overlaid unless the same token is specified.

With CT (create): ESTAE processing places the token created for this request in the location specified by *token addr* as well as in the ESTAE parameter list.

With OV (overlay): ESTAE processing locates the specified ESTAE routine for the current RB and replaces the routine information. If there are any newer ESTAE routines for the RB, they are deleted.

With 0 (cancel): ESTAE processing locates the specified ESTAE routine for the current RB and deletes the routine. Any newer ESTAE routines for the RB are deleted.

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

Control returns to the instruction following the ESTAE macro instruction. When control returns, 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.
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, the most recent exit is not an ESTAE exit, or the ESTAE was created with the TOKEN parameter and on a delete request, either the token was not specified or does not match.
10	An unexpected error was encountered while processing this request.
14	ESTAE was unable to obtain storage for an SCB.
18	The ESTAE was created with the TOKEN parameter and on an overlay request, either the token was not specified or does not match.

Example 1

Operation: If an error occurs, pass control to the ESTAE routine 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
```

Example 2

Operation: If an error occurs, pass control to the ESTAE routine specified by register 4. The address of the ESTAE parameter list is in register 2. Place the token associated with this ESTAE routine in TOKENFLD.

```
ESTAE (4),PARM=(2),TOKEN=TOKENFLD
```

Example 3

Operation: If an error occurs, pass control to the ESTAE routine labeled ADDR, allow synchronous exit processing, halt I/O, allow special error processing, branch enter SVC 60, use the 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 a list form of the macro instruction elsewhere in this program.

```
ESTAE ADDR,ASYNCH=YES,PURGE=HALT,TERM=YES,BRANCH=YES,    X
      SVEAREA=SADDR,MF=(E,EXEC)
```

Example 4

Operation: Request an overlay of the existing ESTAE recovery routine with the following options: the address of the 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.

```
ESTAE   ADDR,OV,PARAM=PLIST,XCTL=YES,PURGE=HALT,ASYNCH=NO
```

Example 5

Operation: Provide the pointer to the recovery code in the register called EXITPTR, place the address of the ESTAE parameter list in register 9. Register 8 points to the area where the ESTAE parameter list (created with the MF=L option) was moved.

```
ESTAE   (EXITPTR),PARAM=(9),MF=(E,(8))
```

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.
b	One or more blanks must precede ESTAE.
ESTAE	
b	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 are explained under the standard form of the ESTAE macro instruction, with the following exception:

,MF=L
specifies the list form of the ESTAE macro instruction.

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. Any combination of exit addr, PARAM=, XCTL=, PURGE=, ASYNCH=, TERM=, RECORD=, and TOKEN= can be specified to dynamically change the contents of the remote ESTAE parameter list. If TOKEN was previously specified and is to be used again without change, TKNPASS=YES must be coded. Any fields not specified on the macro instruction remain as they were 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.
b	One or more blanks must precede ESTAE.
ESTAE	
b	One or more blanks must follow ESTAE.
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=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	
,TOKEN=*token addr*	*token addr:* RX-type address, or register (2) - (12).
,TKNPASS=NO ,TKNPASS=YES	**Default:** TKNPASS=NO
,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 ESTAE macro instruction, with the following exceptions:

,TKNPASS=NO
,TKNPASS=YES
specifies that a previously-specified token, indicated in the parameter list, should be ignored (NO), or should remain part of the specification (YES).

,MF=(E,*ctrl addr***)**
specifies the execute form of the ESTAE macro instruction using a remote control program parameter list.

ETCON - Connect Entry Table

The ETCON macro instruction connects one or more previously created entry tables to the specified linkage table indexes in the current home address space. If an entry table is connected to a system linkage index (an index reserved with the SYSTEM=YES option of the LXRES macro instruction), the entry table is connected to the linkage table of every address space, both present and future.

The restrictions on the use of the ETCON macro instruction are as follows:

- If an entry table contains entries that cause address space switches, the entry table owner must have previously established authorization to issue PT and SSAR instructions to the home address space.
- An entry table can be connected only once to a single linkage table.
- The linkage index and the entry table being connected must be under the same ownership.

Any violation of these restrictions causes the caller to be abnormally terminated.

The connection created by the ETCON macro instruction remains in effect until one of the following occurs:

- The ETDIS macro instruction removes the connection.
- The entry table owner terminates.
- The address space to which the table is connected terminates unless the connection was to a system linkage index.
- The system is re-IPLed.

The caller must be in supervisor state or PKM 0-7, executing in primary mode, enabled, and unlocked. The parameter list passed to the ETCON macro instruction must be addressable in primary mode at the time the macro instruction is issued. Register 13 must point to a standard register save area that must also be addressable in primary mode.

Registers 2-14 are preserved. Register 2, which is modified by the macro after the registers are saved, should not be used as the base register. Register 15 contains the return code. The contents of registers 0 and 1 are unpredictable.

The ETCON macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ETCON.
ETCON	
b	One or more blanks must follow ETCON.
TKLIST=*addr*	*addr:* RX-type address or register (0) - (12).
,LXLIST=*addr*	*addr:* RX-type address or register (0) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

TKLIST = *address*
specifies the address of a list of fullword tokens representing the entry tables to be connected to the linkage table. The first entry in the list must be the number of tokens that follow (from 1 to 32). The tokens are the values returned in register 0 when the ETCRE macro instruction is issued.

,LXLIST = *addr*
specifies the address of a list of linkage index values to which the specified entry tables are to be connected. The list contains fullword entries, the first of which must be the number of linkage index values that follow (from 1 to 32). The number of linkage indexes must be the same as the number of tokens. The first entry table is connected to the first linkage index; the second entry table is connected to the second linkage index, and so on.

,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 can be any valid coding values.

When control returns, register 15 contains the following return code:

Hexadecimal Code	Meaning
0	The specified connections were successfully made.

ETCON (List Form)

The list form of the ETCON macro instruction constructs a non- executable parameter list. This list, or a copy of it for reentrant programs, can be referred to by the execute form of the macro instruction.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ETCON.
ETCON	
b	One or more blanks must follow ETCON.
TKLIST = *addr*	*addr:* A-type address.
,LXLIST = *addr*	*addr:* A-type address.
,RELATED = *value*	*value:* any valid macro keyword specification.
,MF = L	

The parameters are explained under the standard form of the ETCON macro instruction, with the following exception:

,MF = L
specifies the list form of the ETCON macro instruction.

ETCON (Execute Form)

The execute form of the ETCON macro instruction can refer to and modify a remote parameter list created by the list form of the macro.

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

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

TKLIST = *addr*	*addr:* RX-type address or register (0) - (12).
,LXLIST = *addr*	*addr:* RX-type address or register (0) - (12).
,RELATED = *value*	*value:* any valid macro keyword specification.
,MF = *(E,cntl addr)*	*cntl addr:* RX-type address or register (0) - (12).

The parameters are explained under the standard form of the ETCON macro instruction with the following exception:

,MF = (E,*cntl addr*)
specifies the execute form of the ETCON macro instruction. This form uses a remote parameter list.

ETCRE - Create Entry Table

The ETCRE macro instruction causes a program call entry table to be built based upon descriptions of each entry. A token representing the created entry table is returned to the requestor. This token must be used in all subsequent references to the entry table.

The created entry table is owned by the cross memory resource ownership task in the current home address space. When the cross memory resource ownership task terminates, entry tables are disconnected and freed.

The caller must be in supervisor state or PKM 0-7 executing in primary mode enabled and unlocked. Register 13 must point to a standard register save area that must be addressable in primary mode. The list of descriptions specified by ENTRIES must also be addressable in primary mode when the macro instruction is issued.

Registers 2 - 14 are preserved. Register 2, which is modified by the macro after the registers are saved, should not be used as the base register. Register 15 contains the return code. On return, register 0 contains the 32-bit token associated with the new entry table. The contents of register 1 are unpredictable.

The ETCRE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ETCRE.
ETCRE	
b	One or more blanks must follow ETCRE.
ENTRIES = *addr*	*addr:* RX-type address of register (0) - (12).
,RELATED = *value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

ENTRIES = *addr*
specifies the address of the description of the entry table to be built. The entry table description is a table consisting of a single 4 byte table header followed by one 20 byte description element for each entry table entry to be built. The description elements must appear in ascending sequence based on the entry index number. The IHAETD mapping macro defines the format to which the entry table description must conform as shown in Figure 6.

An entry index value that does not have a description results in an invalid entry in the entry table. If the program name field in an entry table description entry contains zeroes, an invalid entry is created for that entry index. A program call to an invalid entry causes the caller to be abnormally terminated. The ETCRE caller is abnormally terminated if any of the reserved fields are nonzero or if the system cannot locate the specified program name.

,RELATED=*value*

specifies information used to self-document macro instructions by relating functions or services to corresponding services performed elsewhere. The format and contents of the information specified can be any valid coding values.

OFFSETS	TYPE	LENGTH	NAME	DESCRIPTION
0 (0)	STRUCTURE	4	ETD	ENTRY TABLE DESCRIPTION LIST DESCRIBES THE INPUT LIST TO THE ETCRE MACRO
0 (0)	UNSIGNED	1	ETDFMT	FORMAT NUMBER MUST BE ZERO
1 (1)	UNSIGNED	1	ETDRSV1	RESERVED MUST BE ZERO
2 (2)	UNSIGNED	2	ETDNUM	NUMBER OF ENTRY DESCRIPTIONS THAT FOLLOW (MAX OF 256)
0 (0)	STRUCTURE	20	ETDELE	ELEMENT DESCRIPTION. ONE FOR EACH ENTRY TO BE ASSIGNED
0 (0)	UNSIGNED	1	ETDEX	INDEX FOR THIS ENTRY (0 ORIGIN)
1 (1)	BITSTRING	1	ETDFLG	FLAG BYTE
	1...		ETDSUP	IF ONE, THE PROGRAM IS TO EXECUTE IN SUPERVISOR STATE, IF ZERO, PROBLEM STATE
	.1..		ETDXM	CROSS MEMORY SPACE SWITCH. IF ZERO THE ENTRY WILL NOT CAUSE A SPACE SWITCH. IF ONE, THE PROGRAM WILL EXECUTE IN THE ADDRESS SPACE OF THE CREATOR OF THE ENTRY TABLE WITH THE AUTHORIZATION OF THAT ADDRESS SPACE.
	..11 1111		ETDRSV2	RESERVED. MUST BE ZERO
2 (2)	UNSIGNED	2	ETDRSV3	RESERVED. MUST BE ZERO
4 (4)	CHARACTER	8	ETDPRO	PROGRAM NAME OR THE VIRTUAL ADDRESS TO BE GIVEN CONTROL. IF A PROGRAM NAME, THE NAMED PROGRAM MUST BE ON THE ACTIVE LPA QUEUE (FLPA OR MLPA) OR BE IN THE PLPA. IF AN ADDRESS, ETDPRO1 MUST BE ZERO AND ETPRO2 MUST BE THE ADDRESS. BIT 0 OF THE ADDRESS FIELD SPECIFIES THE ADDRESSING MODE IN WHICH THE ROUTINE IS TO RECEIVE CONTROL. (IF SET TO 1, THE ADDRESSING MODE IS 31-BIT; IF SET TO 0, THE ADDRESSING MODE IS 24-BIT.
4 (4)	UNSIGNED	4	ETDPRO1	FIRST WORD OF ETDPRO
8 (8)	A-ADDRESS	4	ETDPRO2	SECOND WORD OF ETDPRO

Figure 6 (Part 1 of 2). IHAETD Mapping Macro

```
OFFSETS    TYPE        LENGTH  NAME      DESCRIPTION

 12 (C)    BITSTRING      2    ETDAKM    16 BIT AUTHORIZED KEY MASK.
                                         BIT 0 REPRESENTS KEY 0, ETC.
                                         IF A BIT IS ON, THE
                                         CORRESPONDING KEY IS
                                         AUTHORIZED TO CALL THIS ENTRY
           1... ....                     ETDAK0    BIT REPRESENTING KEY 0
           .1.. ....                     ETDAK1    BIT REPRESENTING KEY 1
           ..1. ....                     ETDAK2    BIT REPRESENTING KEY 2
           ...1 ....                     ETDAK3    BIT REPRESENTING KEY 3
           .... 1...                     ETDAK4    BIT REPRESENTING KEY 4
           .... .1..                     ETDAK5    BIT REPRESENTING KEY 5
           .... ..1.                     ETDAK6    BIT REPRESENTING KEY 6
           .... ...1                     ETDAK7    BIT REPRESENTING KEY 7
-------------------------------------------------------------------------
 13 (D)    1... ....                     ETDAK8    BIT REPRESENTING KEY 8
           .1.. ....                     ETDAK9    BIT REPRESENTING KEY 9
           ..1. ....                     ETDAKA    BIT REPRESENTING KEY 10
           ...1 ....                     ETDAKB    BIT REPRESENTING KEY 11
           .... ....                     ETKAKC    BIT REPRESENTING KEY 12
           .... .1..                     ETDAKD    BIT REPRESENTING KEY 13
           .... ..1.                     ETDAKE    BIT REPRESENTING KEY 14
           .... ...1                     ETDAKF    BIT REPRESENTING KEY 15
 14 (E)    BITSTRING      2    ETDEKM    16 BIT EXECUTION KEY MASK.
                                         BIT 0 REPRESENTING KEY 0, ETC.
                                         IF A BIT IS ON, THE CALLED
                                         PROGRAM IS AUTHORIZED TO
                                         USE THE KEY.
           1... ....                     ETDEK0    BIT REPRESENTING KEY 0
           .1.. ....                     ETDEK1    BIT REPRESENTING KEY 1
           ..1. ....                     ETDEK2    BIT REPRESENTING KEY 2
           ...1 ....                     ETDEK3    BIT REPRESENTING KEY 3
           .... 1...                     ETDEK4    BIT REPRESENTING KEY 4
           .... .1..                     ETDEK5    BIT REPRESENTING KEY 5
           .... ..1.                     ETDEK6    BIT REPRESENTING KEY 6
           .... ...1                     ETDEK7    BIT REPRESENTING KEY 7
 15 (F)    1... ....                     ETDEK8    BIT REPRESENTING KEY 8
           .1.. ....                     ETDEK9    BIT REPRESENTING KEY 9
           ..1. ....                     ETDEKA    BIT REPRESENTING KEY 10
           ...1 ....                     ETDEKB    BIT REPRESENTING KEY 11
           .... 1...                     ETDEKC    BIT REPRESENTING KEY 12
           .... .1..                     ETDEKD    BIT REPRESENTING KEY 13
           .... ..1.                     ETDEKE    BIT REPRESENTING KEY 14
           .... ...1                     ETDEKF    BIT REPRESENTING KEY 15
-------------------------------------------------------------------------
 16 (10)   CHARACTER      4    ETDPAR    PARAMETER TO BE PASSED
                                         TO THE CALLED PROGRAM.
```

Figure 6 (Part 2 of 2). IHAETD Mapping Macro

When control returns, register 15 contains the following return code:

Hexadecimal Code	Meaning
0	The entry table is successfully created.

ETDES - Destroy Entry Table

The ETDES macro instruction destroys a previously-created entry table. Only the address space that owns the entry table can destroy it. At the time ETDES is issued, the entry table must not be connected to any linkage tables unless PURGE=YES is coded. If any outstanding connections still exist and PURGE=YES is not coded, the entry table is not destroyed and the caller is abnormally terminated.

The caller must be in supervisor state or PKM 0-7 executing in primary mode enabled and unlocked. Register 13 must point to a standard register save area the must be addressable in primary mode. The parameter list passed to ETDES must also be addressable in primary mode at the time ETDES is issued.

Registers 2 - 14 are preserved. Register 2, which is modified by the macro after the registers are saved, should not be used as the base register. Register 15 contains the return code. The contents of registers 0 and 1 are unpredictable.

The ETDES macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ETDES.
ETDES	
b	One or more blanks must follow ETDES.
TOKEN=*addr*	*addr:* RX-type address or register (0) - (12).
,PURGE=NO ,PURGE=YES	**Default:** PURGE=NO
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

TOKEN=*addr*
specifies the address of the fullword token (returned by the ETCRE macro instruction) associated with the entry table to be destroyed.

,PURGE=NO
,PURGE=YES
specifies whether (YES) or not (NO) the entry table is to be disconnected from all linkage tables and then destroyed.

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

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

Hexadecimal Code	Meaning
0	The specified entry table was destroyed. There were no connections to linkage indexes.
4	The specified entry table was destroyed. There were connections to linkage indexes, PURGE=YES was specified, and the entry table was disconnected.

ETDES (List Form)

The list form of the ETDES macro instruction constructs a non-executable parameter list. The execute form of the macro can refer to this parameter list, or a copy of it for reentrant programs.

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

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

TOKEN=*addr*	*addr:* A-type address.
,PURGE=NO ,PURGE=YES	**Default:** PURGE=NO
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

The parameters are explained under the standard form of the ETDES macro instruction with the following exception:

,MF=L
specifies the list form of the ETDES macro instruction.

ETDES (Execute Form)

The execute form of the ETDES macro instruction can refer to and modify a remote parameter list created by the list form of the macro.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ETDES.
ETDES	
b	One or more blanks must follow ETDES.
TOKEN=*addr*	*addr:* RX-type address or register (0) - (12).
,PURGE=NO ,PURGE=YES	**Default:** PURGE=NO
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=*(E,cntl addr)*	*cntl addr:* RX-type address or register (0) - (12).

The parameters are explained under the standard form of the ETDES macro instruction with the following exception:

,MF=(E,*cntl addr)*
specifies the execute form of the ETDES macro instruction. This form uses a remote parameter list.

ETDIS - Disconnect Entry Table

The ETDIS macro instruction disconnects one or more entry tables from the home address space's linkage table.

The caller must be in supervisor state or PKM 0-7 executing in primary mode enabled and unlocked. Register 13 must point to a standard register save area that must be addressable in primary mode. The parameter list passed by the requestor must also be addressable in primary mode at the time the macro is issued.

Registers 2-14 are preserved. Register 2, which is modified by the macro after the registers are saved, should not be used as the base register. Register 15 contains the return code. The contents of registers 0 and 1 are unpredictable.

The ETDIS macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede ETDIS.
ETDIS	
b	One or more blanks must follow ETDIS.
TKLIST=*addr*	*addr:* RX-type address or register (0) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

TKLIST=*addr*
specifies the address of a list of 1 to 32 fullword tokens, returned by the ETCRE macro instruction, identifying the entry tables to be disconnected from the home address space's linkage table. The first entry of the list must be a fullword count of the number of tokens (1 to 32) in the list.

,RELATED=*value*
specifies information used to self-document macro instructions by "relating" functions or services to corresponding services performed elsewhere. The format and contents of the information specified can be any valid coding values.

When control returns, register 15 contains the following return code:

Hexadecimal Code	Meaning
0	The entry table is successfully disconnected.

EVENTS - Wait for One or More Events to Complete

This macro can be assembled compatibly between MVS/XA and MVS/370 through the use of the SPLEVEL macro instruction. Default processing will result in an expansion of the macro that operates only with MVS/XA. See the topic "Selecting the Macro Level" for additional information. If you are executing in 31-bit addressing mode, you must use the MVS/XA version of this macro instruction.

The EVENTS macro instruction is a functional specialization of the WAIT ECBLIST= macro facility with the advantages of notifying the program that events have completed and the order in which they completed.

The macro performs the following functions:

- Creates and deletes EVENTS tables.
- Initializes and maintains a list of completed event control blocks.
- Provides for single or multiple ECB processing.

The description of the EVENTS macro instruction follows. The EVENTS macro instruction is also described in *Supervisor Services and Macro Instructions* with the exception of the BRANCH parameter. This parameter is restricted to programs that run in supervisor state, key 0, and own the LOCAL lock.

Note: CML (cross memory local) lock means the local lock of an address space other than the home address space. LOCAL lock means the local lock of the home address space. When written in lower case, local lock means any local-level lock, either the LOCAL or a CML lock.

The EVENTS macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede EVENTS.
EVENTS	
b	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 are explained below:

ENTRIES = *n*

n is a decimal number from 1 to 32,767 which specifies the maximum number of completed ECB addresses that can be processed in an EVENTS table concurrently.

Note: When this parameter is specified, no other parameter should be specified.

ENTRIES = DEL, TABLE = *tab addr*

specifies that the EVENTS table whose address is specified by TABLE = *tab addr* is to be deleted. The user is responsible for deleting all of the tables he creates; however, all existing tables are automatically freed at task termination.

Notes:

1. *When this parameter is specified, no other parameter should be specified.*
2. *TABLE resides in 24-bit addressable storage.*

TABLE = *tab addr*

specifies either a register number or the address of a word containing the address of the EVENTS table associated with the request. The address specified with the operand TABLE must be that of an EVENTS table created by this task.

Note: TABLE resides in 24-bit addressable storage.

,WAIT=NO
,WAIT=YES

specifies whether or not to put the issuing program in a wait state when there are no completed events in the EVENTS TABLE (specified by the TABLE= parameter).

,ECB=*ecb addr*

specifies either a register number or the address of a word containing the address of an event control block. The EVENTS macro instruction should be used to initialize any event-type ECB. To avoid the accidental destruction of bit settings by a system service such as an access method, the ECB should be initialized after the system service that will post the ECB has been initiated (thus making the ECB eligible for posting) and before the EVENTS macro is issued to wait on the EVENTS table.

Notes:

1. *Register 1 should not be specified for the ECB address.*
2. *This parameter may not be specified with the LAST= parameter.*
3. *The ECB can reside above or below 16 megabytes.*
4. *If only ECB initialization is being requested, neither WAIT=NO nor WAIT=YES should be specified, to prevent any unnecessary WAIT processing from occurring.*

,LAST=*last addr*

specifies either a register number or the address of a word containing the address of the last EVENT parameter list entry processed.

Notes:

1. *Register 1 should not be specified for the LAST address.*
2. *This parameter should not be specified with the ECB= parameter.*
3. *The WAIT macro must also be specified.*
4. *LAST resides in 24-bit addressable storage.*

,BRANCH=NO
,BRANCH=YES

specifies that an SVC entry (BRANCH=NO) or a branch entry (BRANCH=YES) is to be performed.

Example 1

The following shows total processing via EVENTS

EVENTS and ECB Initialization

```
EVENTS     ENTRIES=1000

ST         R1,TABADD

WRITE      ECBA

LA         R2,ECBA...

EVENTS     TABLE=TABADD,ECB=(R2)
```

Parameter List Processing

```
         EVENTS     TABLE=TABADD,WAIT=YES

         LR         R3,R1         PARMLIST ADDR
         B          LOOP2         GO TO PROCESS ECB
LOOP1    EVENTS     TABLE=TABADD,WAIT=YES,LAST=(R3)
         LR         R3,R1         SAVE POINTER
LOOP2    EQU        *             PROCESS COMPLETED EVENTS

         TM         0(R3),X'80'   TEST FOR MORE EVENTS
         BO         LOOP1         IF NONE, GO WAIT
         LA         R3,4(,R3)     GET NEXT ENTRY
         B          LOOP2         GO PROCESS NEXT ENTRY
```

Deleting EVENTS Table

```
         EVENTS     TABLE=TABADD,ENTRIES=DEL

TABADD   DS         F
```

Example 2

Processing One ECB at a Time.

```
                    EVENTS      ENTRIES=10
                    ST          1,TABLE

         NEXTREC    GET         TPDATA,KEY
                    ENQ         (RESOURCE,ELEMENT,E,,SYSTEM)
                    READ        DECBRW,KU,,'S',MF=E
                    LA          3,DECBRW
                    EVENTS      TABLE=TABLE,ECB=(3),WAIT=YES

                    WRITE       DECBRW,K,MF=E
                    LA          3,DECBRW
         RETEST     EVENTS      TABLE=TABLE,ECB=(3),WAIT=NO
                    LTR         1,1
                    BNZ         NEXTREC

                    B           RETEST

         TABLE      DS          F
```

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 control program places the information in an area that the problem program provides. For a description of this area see "Providing an EXTRACT Answer Area" in the Subtask Creation and Control section in Volume 1. When the extract macro is issued, its parameter list must reside in 24-bit addressable storage.

Notes:

1. *If the EXTRACT macro is used to obtain the TIOT in order to find the UCB, it is the user's responsibility to ensure that the TIOT contains the UCB address. To find the UCB address, refer to the topic "Finding the UCB Address" in Volume 1.*

2. *Because the parameter list for EXTRACT must be in 24-bit addressable storage, the standard form of the EXTRACT macro can only be issued by programs residing in 24-bit addressable storage*

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede EXTRACT.
EXTRACT	
b	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 as follows:

answer addr
specifies the address of the answer area to contain the requested information. The area is one or more fullwords, starting on a fullword boundary. The number of fullwords must be the same as the number of fields specified in the FIELDS parameter, unless ALL is coded. If ALL is coded, seven fullwords are required.

Note: answer addr resides in 24-bit addressable storage.

,'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 is the default, no address is specified and the active task is assumed.

Note: The TCB address resides in 24-bit addressable storage.

,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, and a token. (If a token exists, the high order bit of the token field is set to one). The token is used only with internal START commands. See "Issuing an Internal Start Command" in Volume 1.
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 TSO protected step 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 that 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)
```

Example 3

Operation: Provide information from the command scheduler communications list. ANSWER is the label of the answer area and TCBADDR is the label of a fullword that contains the address of the subtask TCB from which information is to be extracted.

```
EXTRACT ANSWER,TCBADDR,FIELDS=(COMM)
```

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.
b	One or more blanks must precede EXTRACT.
EXTRACT	
b	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 exception:

,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 canceled 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.
b	One or more blanks must precede EXTRACT.
EXTRACT	
b	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 TIOT 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 exception:

,MF = (E, *ctrl addr)*
specifies the execute form of the EXTRACT macro instruction using a remote control program parameter list.

Note: The parameter list must reside in 24-bit addressable storage.

FESTAE - Fast Extended STAE

This macro can be assembled compatibly between MVS/XA and MVS/370 through the use of the SPLEVEL macro instruction. Default processing will result in an expansion of the macro that operates only with MVS/XA. See the topic "Selecting the Macro Level" for additional information.

Except for the TCB, all input parameters to this macro instruction can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

FESTAE users executing in 31-bit addressing mode (the A bit in the PSW is on) must recompile using the MVS/XA FESTAE macro expansion so that the exit routine gets control in 31-bit addressing mode.

The FESTAE macro instruction allows an SVC to establish an ESTAE recovery routine with minimal overhead and no locking requirements. The ESTAE routine activated by FESTAE receives control in the same sequence and under the same conditions as though created by the ESTAE macro instruction. The FESTAE macro instruction can be issued in cross memory mode as long as the currently addressable address space is the home address space. The interface to the FESTAE exit is described in Volume 1 under "Using the FESTAE Macro Instruction."

The FESTAE macro expansion has no external linkage. The macro instruction is written as follows:

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

0,WRKREG=*work reg addr* EXITADR=*exit addr*	*work reg addr:* Register (1) - (14). *exit addr:* Register (1) - (14).
,RBADDR=*svrb addr*	*svrb addr:* Register (1) - (14).
,TCBADDR=*tcb addr*	*tcb addr:* Register (1) - (14).
,PARAM=*list addr*	*list addr:* Register (1) - (14).
,XCTL=NO ,XCTL=YES	**Default:** XCTL=NO
,PURGE=NONE ,PURGE=HALT ,PURGE=QUIESCE	**Default:** PURGE=NONE
,ASYNCH=YES ,ASYNCH=NO	**Default:** ASYNCH=YES
,TERM=NO ,TERM=YES	**Default:** TERM=NO
,RECORD=NO ,RECORD=YES	**Default:** RECORD=NO
,ERRET=*label*	*label:* any valid assembler name.

The parameters are explained as follows:

0,WRKREG=*work reg addr*
specifies that the current ESTAE routine be canceled if it was created by FESTAE. An error occurs if the current ESTAE routine was not created by FESTAE. A work register must be specified for use by the FESTAE macro expansion.

,EXITADR=*exit addr*
specifies a register that contains the address of an ESTAE routine to be entered if the task terminates abnormally. This register is used subsequently as a work register.

,RBADDR=*svrb addr*
specifies a register that contains the address of the current SVRB prefix (RBPRFX). RBADDR must be specified if EXITADR has also been specified. The specified register is not altered.

,TCBADDR = *tcb addr*
specifies the register containing the current TCB address. This register is not altered, and its specification results in the generation of more efficient code.

Note: The TCB resides in storage below 16 megabytes.

,PARAM = *list addr*
specifies the register containing the address of a user-defined parameter list that contains data to be used by the ESTAE routine. The routine receives this address when it is scheduled for execution.

The use of this parameter list is optional, but the user should zero out any spurious data it might contain whether or not he intends to use it. If the user does not select the PARAM option, the routine receives instead the 24-byte parameter area in the SVRB. In this case, the user must locate this SVRB parameter area and initialize it with appropriate data.

,ERRET = *label*
specifies a label within the CSECT issuing the FESTAE for which addressability has been established. The FESTAE macro instruction branches to this label if it is returning a code other than zero. This option saves the user the instructions necessary to check the return code. If the user does not specify the ERRET option, control returns instead to the instruction immediately following the FESTAE macro instruction. The return code is in register 15.

All the other FESTAE parameters have the same meaning as their ESTAE counterparts.

Upon conclusion of FESTAE processing, control resumes at the instruction following the FESTAE macro instruction. Register 15 then contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful completion of the FESTAE request.
08	A previous create has been issued with FESTAE for this SVRB; the request has been ignored.
0C	Cancel has been specified under one of the following conditions: 1) There is no exit for this TCB. 2) The most recent exit is not owned by the caller. 3) The most recent exit was not created by FESTAE.

Example 1

Operation: In case of an abnormal termination, execute the ESTAE routine specified by register 2, allow asynchronous processing, do not allow special error processing, default to PURGE=NONE, and pass the parameter list pointed to by register 7 to the ESTAE routine.

```
FESTAE   EXITADR=(REG2),RBADDR=(REG3),TCBADDR=(REG6),    X
         PARAM=(REG7),ASYNCH=YES,TERM=NO
```

FREEMAIN - Free Virtual Storage

The FREEMAIN macro instruction releases one or more areas of virtual storage, or an entire virtual storage subpool, previously assigned to the active task as a result of a GETMAIN macro instruction. FREEMAIN is supported in a cross memory environment. The active task is abnormally terminated if the specified virtual storage does not start on a doubleword boundary or, for an unconditional request, if the specified area or subpool is not currently allocated to the active task. Register 15 is set to 0 to indicate successful completion. For a conditional FREEMAIN, register 15 is set to 4 if the specified area or subpool is not currently allocated to the active task.

In the parameters discussed below, EU, LU, and VU specify unconditional requests and result in the same processing as E, L, and V, respectively. The two formats for these requests are available to maintain compatibility with the GETMAIN formats. Users of the FREEMAIN macro instruction who are freeing virtual storage with addresses greater than 16 megabytes must use either the RC or RU form of the macro instruction. All addresses specified with the RC or RU form of the macro are treated as 31-bit addresses.

The description of the FREEMAIN macro instruction follows. The FREEMAIN macro instruction is also described in *Supervisor Services and Macro Instructions* with the exception of the BRANCH and KEY parameters. These parameters are restricted to programs running supervisor state, key 0 and, therefore, are only described here.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede FREEMAIN.
FREEMAIN	
b	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, or register (0) or (2) - (12). **Notes:** 1. If R,SP=(0) is specified with no other parameters, the high order byte of register 0 must contain the subpool number and the low order 3 bytes must contain zero. 2. For a subpool FREEMAIN, if RC,SP=*subpool numbr* or RU,SP=*subpool nmbr* or R,SP=*subpool nmbr* is specified, no other parameters except RELATED can be specified. 3. RC and RU are the only parameters that can be used to free storage above 16 megabytes.
,A=*addr*	*addr:* A-type address, or register (2) - (12). **Note:** If R, RC, or RU is coded, register (1) can also be specified.
,SP=*subpool nmbr*	*subpool nmbr:* symbol, decimal digit, or register (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 be specified with RC or RU above. Also, the macro expansion uses register 4 for the address of the global save area pointed to by the CVT. The previous contents of register 4 is overridden.
,KEY=*nmbr*	*nmbr:* decimal digits 0-15, or register (2) - (12). **Note:** This parameter may be specified only with BRANCH and RC or RU above.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained below:

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*

specifies the type of FREEMAIN request:

LC, LU, and L indicate conditional (LC) and unconditional (LU and L) list requests and specify release of one or more areas of virtual storage. The length of each virtual storage area is indicated by the values in a list beginning at the address specified in the LA parameter. The address of each of the virtual storage areas must be provided in a corresponding list whose address is specified in the A parameter. All virtual storage areas must start on a doubleword boundary.

VC, VU, and V indicate conditional (VC) and unconditional (VU and V) variable requests and specify release of single areas of virtual storage. The address and length of the virtual storage area are provided at the address specified in the A parameter.

EC, EU, and E indicate conditional (EC) and unconditional (EU and E) element requests and specify release of single areas of virtual storage. The length of the single virtual storage area is indicated in the LV parameter. The address of the virtual storage area is provided at the address indicated in the A parameter.

RC, RU, and R indicate conditional (RC) and unconditional (RU and R) register requests and specify release of single areas of virtual storage from the subpool indicated, or specifies release of the entire subpool indicated. If the release is not for the entire subpool, the address of the virtual storage area is indicated in the A parameter. The length of the area is indicated in the LV parameter. The virtual storage area must start on a doubleword boundary.

Notes:

1. *A conditional request indicates that the task is not to be abnormally terminated if the virtual storage being freed is not allocated to the active task. However some abends cannot be prevented. An unconditional request indicates that the task is to be abnormally terminated in this situation.*

2. *Callers in either 24-bit or 31-bit addressing mode can use RC or RU to free storage above 16 megabytes.*

3. *If the address of the area to be freed is greater than 16 megabytes, you must use RC or RU.*

LA specifies the virtual storage address of one or more consecutive fullwords starting on a fullword boundary. One word is required for each virtual storage area to be released; the high-order bit in the last word must be set to 1 to indicate the end of the list. Each word must contain the required length in the low-order three bytes. The fullwords in this list must correspond with the fullwords in the associated list specified in the A parameter. The words must not be in the area to be released. If this rule is violated and if the words are the last allocated items on a virtual page, the whole page is returned to storage and the FREEMAIN abends with an 0C4.

LV specifies the length, in bytes, of the virtual storage area being released. The value should be a multiple of 8; if it is not, the control program uses the next high multiple of 8. If R is coded, LV=(0) may be designated; the high-order byte of register 0 must contain the subpool number, and the low-order three bytes must contain the length (in this case, the SP parameter is invalid).

,A=*addr*

specifies the virtual storage address of one or more consecutive fullwords starting on a fullword boundary. The input should not reside within the area to be released. If this rule is violated and if the input is within the area and is the last allocated item on a virtual page, the whole page is returned to storage and the FREEMAIN abends with an 0C4. If E, EC, EU, R, RC, or RU is designated, one word, which contains the address of the virtual storage area to be released, is required. If V, VC, or VU is coded, two words are required; the first word contains the address of the virtual storage area to be released, and the second word contains the length of the area to be released. If L, LC, or LU is coded, one word is required for each virtual storage area to be released; each word contains the address of one virtual storage area. If R, RC, or RU is coded, any of the registers 1 through 12 can be designated, in which case the address of the virtual storage area, not the address of the fullword, must have previously been loaded into the register.

,SP=*subpool nmbr*

specifies the subpool number of the virtual area to be released. The subpool number can be between 0 and 255. The SP parameter is optional and if omitted, subpool 0 is assumed. If R is coded, SP=(0) can be designated, in which case the subpool number must be previously loaded into the low-order byte of register 0.

For subpool freemains, the SP parameter specifies the number of the subpool to be released. Subpool freemains can be issued only for the following subpools: 1-127, 229, 230, 233, 236, 237, 240, and 250-253; and if the caller is in key 0, subpool 0. Any attempt to issue a subpool freemain for any other subpool causes a 478 or 40A abend. (See Volume 1 for a list of the characteristics of the valid subpools.) If R,SP=(0) is specified with no other parameters, the high-order byte byte of register 0 must contain the subpool number and the low-order 3 bytes must contain zero.

,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 entry point to service global storage requests without the need for the local address space lock will be used. However, the caller must not hold any lock higher than the VSMFIX lock (for subpools 226, 227, 228, 239, and 245) or the VSMPAG lock (for subpools 231 and 241) and the caller must be disabled.

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 of the

ASCB address specified in register 7 prior to entry. Register 3 will be destroyed if RC or RU was specified.

Callers in cross memory mode can use the BRANCH=YES parameter of the FREEMAIN macro instruction. If the caller is in cross memory mode, the storage is freed in the currently addressable address space. The caller must hold the CML lock for the currently addressable address space; load register 7 with the address of the ASCB of the currently addressable address space; and load register 4 with zero or the address of a TCB in the currently addressable address space. If register 4 contains a zero, the storage that is freed is associated with the current job step task that owns the cross memory resources in the currently addressable address space (that is, the TCB anchored in ASCBXTCB).

If BRANCH=(YES,GLOBAL) is specified, registers 4 and 7 need not contain the TCB and ASCB addresses; and registers 3 and 4 are changed when control is returned to the caller. Additionally, the SP parameter may only designate subpools 226,227, 228, 231, 239, 241, or 245.

,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 specified storage protection key.

,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 can be any valid coding values.

When control is returned, register 15 contains one of the following return codes. A code other than 0 is possible only for conditional forms.

Hexadecimal Code	Meaning
00	Virtual storage was freed.
04	Not all virtual storage was freed.
08	Part of area being freed is still fixed. This condition causes an A78, A05, or A0A abend unless the TCBEOTFM indicator is on.
12	Page table is paged out.

Example 1

Operation: Free 400 bytes of storage addressed by register 2 via a branch entry. If the storage is successfully freed, register 15 contains 0; otherwise, register 15 contains a nonzero value.

```
FREEMAIN EC,LV=400,A=(2),BRANCH=YES
```

Example 2

Operation: Free 48 bytes of the storage (addressed by register 5) in subpool 231. Register 3 has been preset to contain the storage key of the storage to be released. If the request is unsuccessful, the caller is abnormally terminated.

```
FREEMAIN RU,LV=48,A=(5),SP=231,KEY=(3),BRANCH=(YES,GLOBAL)
```

FREEMAIN (List Form)

Use the list form of the FREEMAIN macro instruction to construct a nonexecutable control program parameter list.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede FREEMAIN.
FREEMAIN	
b	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. **Notes:** **1.** LA may only be specified with LC, LU, or L above. **2.** 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.
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

The parameters are explained under the standard form of the FREEMAIN macro instruction, with the following exceptions:

,MF=L
specifies the list form of the FREEMAIN macro instruction.

FREEMAIN (Execute Form)

A remote control program parameter list is used in, and can be modified by, the execute form of the FREEMAIN macro instruction. The parameter list can be generated by the list form of either a GETMAIN or a FREEMAIN.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede FREEMAIN.
FREEMAIN	
b	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). **Notes:** **1.** LA may only be specified with LC, LU, or L above. **2.** 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).

The parameters are explained under the standard form of the FREEMAIN macro instruction, with the following exceptions:

,MF = (E,*ctrl prog*)
specifies the execute form of the FREEMAIN macro instruction using a remote control program parameter list.

GETMAIN - Allocate Virtual Storage

The GETMAIN macro instruction requests the control program to allocate one or more areas of virtual storage to the active task. For task related subpools, the virtual storage areas are allocated from the specified subpool in the virtual storage area assigned to the associated job step. The virtual storage areas each begin on a doubleword or page boundary and are not cleared to 0 when allocated. (The storage is zeroed for the first allocation of a page.) The total of the lengths specified must not exceed the length available. For most subpools, the storage will be released when the task assigned ownership terminates, or through the use of the FREEMAIN macro instruction. For information on when storage that is obtained with a GETMAIN macro is released, see *MVS/XA System Logic Library: Virtual Storage Management*, LY28-1790.

The options R, LC, LU, VC, VU, EC, or EU can be used by callers in either 24-bit or 31-bit addressing mode. If one of these options is specified, storage area addresses and lengths will be treated as 24-bit addresses and values. The parameter list addresses and the pointers to the length and address lists in the parameter lists (if present) will be treated as 31-bit addresses if the caller's addressing mode is 31-bit; otherwise, they will be treated as 24-bit addresses.

The options RU, RC, VRU, and VRC can be used by callers in either 24-bit or 31-bit addressing mode. However, all values and addresses will be treated as 31-bit values and addresses. The GETMAIN macro is also described in *Supervisor Service and Macro Instructions* with the exception of the BRANCH and KEY parameters. These parameters are restricted in use to programs running supervisor state, key 0.

The description of the GETMAIN macro instruction follows.

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede GETMAIN.
GETMAIN	
b	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, RC, or RU is specified, register (0) may also be specified. *addr:* A-type address, or register (2) - (12). **Note:** RC, RU, VRC, or VRU must be used for addresses greater than 16 megabytes.
VRC,LV = *(maximum length value, minimum length value)* VRU,LV = *(maximum length value, minimum length value)*	*maximum length value:* symbol, decimal, digit, or register (2) - (12). *minimum length value:* symbol, decimal, digit, or register (2) - (12).
,SP = *subpool nmbr*	*subpool nmbr:* symbol, decimal digit, or register (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, RU, VRC, or VRU above. Also, the macro expansion uses register 4 for the address of the global save area pointed to by the CVT. The previous contents of register 4 is overridden. The macro expansion also uses register 3.
,KEY = *key number*	*key number:* decimal digits 0-15, or register (2) - (12). **Default:** KEY = 0 **Note:** This parameter may be specified only with BRANCH and RC, RU, VRC, or VRU; and subpools 226, 227, 228, 229, 230, 231, and 241.
,LOC = BELOW ,LOC = (BELOW,ANY) ,LOC = (ANY) ,LOC = (ANY,ANY) ,LOC = RES ,LOC = (RES,ANY)	**Default:** LOC = RES **Note:** This parameter can only be used with RC, RU, VRC, or VRU. On all other forms, LOC = BELOW is used.
,RELATED = *value*	*value:* any valid macro keyword specification.

The parameters are explained below:

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*
VRC,LV = *(maximum length value,minimum length value)*
VRU,LV = *(maximum length value,minimum length value)*
specifies the type of GETMAIN request:

LC and LU indicate conditional (LC) and unconditional (LU) list requests, and specify requests for one or more areas of virtual storage. The length of each virtual storage area is indicated by the values in a list beginning at the address specified in the LA parameter. The address of each of the virtual storage areas is returned in a list beginning at the address specified in the A parameter. No virtual storage is allocated unless all of the requests in the list can be satisfied.

VC and VU indicate conditional (VC) and unconditional (VU) variable requests, and specify requests for single areas of virtual storage. The length of the single virtual storage area is between the two values at the address specified in the LA parameter. The address and actual length of the allocated virtual storage area are returned by the control program at the address indicated in the A parameter.

EC and EU indicate conditional (EC) and unconditional (EU) element requests, and specify requests for single areas of virtual storage. The length of the single virtual storage area is indicated by the parameter, LV = *length value*. The address of the allocated virtual storage area is returned at the address indicated in the A parameter.

RC indicates a conditional register request, and RU and R indicate unconditional register requests. RC, RU, and R specify requests for single areas of virtual storage. The length of the single virtual area is indicated by the parameter, LV = *length value*. The address of the allocated virtual storage area is returned in register 1.

VRC and VRU indicate variable register conditional (VRC) and unconditional (VRU) requests for a single area of virtual storage. The length returned will be between the maximum and minimum lengths specified by the parameter LV = (*maximum length value, minimum length value*). The address of the allocated virtual storage is returned in register 1 and the length in register 0.

Notes:

1. *A conditional request indicates that the task is not to be abnormally terminated if virtual storage is not allocated to the active task. An unconditional request indicates that the task is to be abnormally terminated in this situation.*

2. *The LC, LU, VC, VU, EC, EU, and R forms of the GETMAIN macro instruction can only be used to obtain virtual storage with addresses below 16 megabytes. The RC, RU, VRC, and VRU forms of the GETMAIN macro instruction can be used to obtain virtual storage with addresses above 16 megabytes.*

LA specifies the virtual storage address of consecutive fullwords starting on a fullword boundary. Each fullword must contain the required length in the low-order three bytes, with the high-order byte set to 0. The lengths should be multiples of 8; if they are not, the control program uses the next higher multiple of 8. If VC or VU was coded, two words are required. The first word contains the minimum length required, the second word contains the maximum length. If LC or LU was coded, one word is required for each virtual storage area requested; the high-order bit of the last word must be set to 1 to indicate the end of the list. The list must not overlap the virtual storage area specified in the A parameter.

LV=*length value* specifies the length, in bytes, of the requested virtual storage. The number should be a multiple of 8; if it is not, the control program uses the next higher multiple of 8. If R is specified, LV=(0) may be coded; the low-order three bytes of register 0 must contain the length, and the high-order byte must contain the subpool number. LV=(*maximum length value, minimum length value*) specifies the maximum and minimum values of the length of the storage request.

The A parameter specifies the virtual storage address of consecutive fullwords, starting on a fullword boundary. The control program places the address of the virtual storage area allocated in one or more words. If E was coded, one word is required. If LC or LU was coded, one word is required for each entry in the LA list. If VC or VU was coded, two words are required. The first word contains the address of the virtual storage area, and the second word contains the length actually allocated. The list must not overlap the virtual storage area specified in the LA parameter.

,SP=*subpool nmbr*
specifies the number of the subpool from which the virtual storage area is to be allocated. The subpool number must be a valid subpool number between 0 and 255. See "Virtual Storage Management" in Volume 1 for a list of the valid subpools. If this parameter is omitted, subpool 0 is assumed.

Note: Callers executing in supervisor state and key zero, who specify subpool 0, will obtain storage from subpool 252. Therefore, when requesting a dump of this storage via the SDUMP macro instruction, they must specify subpool 252 rather than 0.

,BNDRY=DBLWD
,BNDRY=PAGE
specifies that alignment on a doubleword boundary (DBLWD) or alignment with the start of a virtual page on a 4K boundary (PAGE) is required for the start of a requested area.

The BNDRY=PAGE keyword is ignored if the request specifies one of the following subpools 226, 233-235, 239,245, or 253-255. Requests for storage from these subpools are fulfilled from a single page, unless the request is greater than a page.

,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 entry point to service global storage requests without the need for the **local** lock is used. The caller must be disabled by obtaining the CPU lock to provide system-recognized disablement and must not hold any lock that would prevent VSM from obtaining the VSMFIX lock (for subpools 226, 227, 228, 239, and 245) or the VSMPAG lock (for subpools 231 and 241). 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** lock prior to entry. The contents of register 3 is destroyed if RC, RU, VRC, or VRU is specified.

Callers in cross memory mode can use the BRANCH=YES parameter of the GETMAIN macro instruction. If the caller is in cross memory mode, the storage is allocated in the currently addressable address space. The caller must hold the CML lock for the currently addressable address space; load register 7 with the address of the ASCB of the currently addressable address space; and load register 4 with zero or the address of a TCB in the currently addressable address space. If register 4 contains a zero, the allocated storage is associated with the current job step task that owns the cross memory resources in the currently addressable address space (that is, the TCB anchored in ASCBXTCB).

If BRANCH=(YES,GLOBAL) is specified, registers 4 and 7 need not contain the TCB and ASCB addresses; and registers 3 and 4 are changed when control returns to the caller. The caller must be disabled and must not hold any locks that would prevent VSM from obtaining the VSMFIX lock (for subpools 226, 227, 228, 239, and 245) or the VSMPAG lock (for subpools 231 and 241). Additionally, the SP parameter may only designate subpools 226, 227, 228, 231, 239, 241, or 245.

,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 specified storage protection key. The KEY parameter cannot be specified unless the BRANCH parameter is also specified.

,LOC=BELOW
,LOC=(BELOW,ANY)
,LOC=ANY
,LOC=(ANY,ANY)
,LOC=RES
,LOC=(RES,ANY)
specifies the location of virtual storage and real storage. This is especially helpful for callers with 24-bit dependencies. When LOC is specified, real storage is allocated anywhere until the storage is fixed (by the PGFIX, PGFIXA, or PGSER macro instructions). After the storage is fixed, virtual and real storage are located in the following manner.

LOC=BELOW indicates that real and virtual storage are to be located below 16 megabytes.

LOC=(BELOW,ANY) indicates that virtual storage is to be located below 16 megabytes and real storage can be located anywhere.

LOC=ANY and LOC=(ANY,ANY) indicate that virtual and real storage can be located anywhere.

Note: The LOC parameter is not valid for fixed subpools. For fixed subpools the actual location of the virtual storage area depends on the subpool specified. If the subpool is supported (backed) above 16 megabytes, GETMAIN attempts to locate the virtual storage area above 16 megabytes. If this is not possible, GETMAIN locates the virtual storage below 16 megabytes. If the subpool is not supported above 16 megabytes, GETMAIN also locates the virtual storage below 16 megabytes. See "Virtual Storage Management" in Volume 1 for a list of valid subpools and their characteristics. For example, LSQA subpools will be backed anywhere regardless of the LOC parameter specified.

LOC=RES indicates that the location of virtual and real storage depends on the location of the caller. If the caller resides below 16 megabytes, virtual and real storage are to be located below 16 megabytes; if the caller resides above 16 megabytes, virtual and real storage are to be located anywhere.

LOC=(RES,ANY) indicates that the location of virtual storage depends upon the location of the caller. If the caller resides below 16 megabytes, virtual storage is to be located below 16 megabytes; if the caller resides above 16 megabytes, virtual storage can be located anywhere. In either case, real storage can be located anywhere.

,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.

When control is returned on conditional type requests (LC, EC, VC, RC, VRC), register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Virtual storage requested was allocated
04	No virtual storage was allocated
08	Real storage was not available for backing the request or internal storage management control blocks.

The contents of registers 0, 1, and 15 are not preserved when the GETMAIN macro instruction is issued.

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 is used; otherwise, subpool 0 is used. If the storage cannot be obtained, the caller is 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) is returned.

```
GETMAIN RC,LV=4096,SP=231,BRANCH=(YES,GLOBAL),BNDRY=PAGE,KEY=9
```

Example 3

Operation: Obtain 400 bytes of storage from subpool 10. If the storage is available, the address will be returned in register 1 and register 15 will contain 0; if storage is not available, register 15 will contain 4.

```
GETMAIN   RC,LV=400,SP=10
```

Example 4

Operation: Obtain 48 bytes of storage from default subpool 0. If the storage is available, the address will be stored in the word at AREAADDR; if the storage is not available, the task will be abnormally terminated.

```
          GETMAIN EU,LV=48,A=AREAADDR
          .
          .
          .
AREAADDR  DS      F
```

Example 5

Operation: Obtain a maximum of 4096 or a minimum of 1024 bytes of virtual storage, with addresses above or below 16 megabytes. Indicate that if the real storage is fixed, it should also be located above or below 16 megabytes. If the storage is available, the address will be returned in register 1 and the length of the storage allocated will be returned in register 0; if the storage is not available, the task will be terminated.

```
GETMAIN VRU,LV=(4096,1024),LOC=ANY
```

GETMAIN (List Form)

Use the list form of the GETMAIN macro instruction to construct a control program parameter list. The list form of the GETMAIN macro instruction cannot be used to allocate virtual storage with addresses greater than 16 megabytes.

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

name	*name:* Begin *name* in column 1.
b	One or more blanks must precede GETMAIN.
GETMAIN	
b	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. **Notes:** **1.** LA may be specified with EC or EU above. **2.** 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.
,BNDRY=DBLWD ,BNDRY=PAGE	**Default:** BNDRY=DBLWD
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

The parameters are explained under the standard form of the GETMAIN macro instruction, with the following exception:

,MF=L
specifies the list form of the GETMAIN macro instruction.

GETMAIN (Execute Form)

A remote control program parameter list is used in, and can be modified by, the execute form of the GETMAIN macro instruction. The parameter list can be generated by the list form of either a GETMAIN or a FREEMAIN. The execute form of the GETMAIN macro instruction cannot be used to allocate virtual storage with addresses greater than 16 megabytes.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede GETMAIN.
GETMAIN	
b	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, 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 are explained under the standard form of the GETMAIN macro instruction, with the following exception:

,MF=(E,*ctrl prog***)**
specifies the execute form of the GETMAIN macro instruction using a remote control program parameter list.

GQSCAN - Extract Information From Global Resource Serialization Queue

Use the GQSCAN macro instruction to obtain the status of resources and requestors of those resources. The GQSCAN macro instruction, in conjunction with the ISGRIB mapping macro, enables you to obtain resource information from system control blocks without knowing the exact structure or location of the control blocks.

The issuer of the GQSCAN macro instruction must be executing in primary mode. To use SCOPE=GLOBAL and SCOPE=LOCAL, you must be in supervisor state or key 0 and you should be aware that improper use of these parameters degrades system performance.

Global resource serialization counts and limits the number of outstanding global resource serialization requests. A global resource serialization request is any ENQ, RESERVE, or GQSCAN that causes an element to be inserted into a queue in the global resource serialization request queue area. See "Limiting Global Resource Serialization Requests" in Volume 1.

Register 13 must contain the address of an 18-word save area, which can be provided through the use of standard linkage conventions.

On return, register 0 contains two halfword values. The first (high order) halfword contains the length of the fixed portion of each RIB returned; the second (low order) halfword contains the length of each RIBE returned. Register 1 contains the number of RIBs that were copied into the area provided. Register 15 contains the return code. In order to interpret the data that the GQSCAN service routine returns in the user-specified area, you must include the ISGRIB mapping macro as a DSECT in your program. ISGRIB maps the resource information block (RIB) and the resource information block extent (RIBE) as shown in the *Debugging Handbook*.

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

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

AREA=*(area addr,area size)*	*area addr:* RX-type address or register (2) - (12). *area size:* symbol, decimal digit, or register (2) - (12). **Note:** AREA cannot be specified with QUIT=YES.
,REQLIM=*value* ,REQLIM=MAX	*value:* symbol, decimal digit, register (2) - (12), or the word MAX. **Default:** REQLIM=MAX
,SCOPE=ALL ,SCOPE=STEP ,SCOPE=SYSTEM ,SCOPE=SYSTEMS ,SCOPE=LOCAL ,SCOPE=GLOBAL	**Default:** SCOPE=STEP
,RESERVE=YES ,RESERVE=NO	**Default:** All resources requested with RESERVE and all resources requested with ENQ.
,RESNAME=*(qname addr[,rname addr, rname length], [GENERIC|SPECIFIC], qname length)*	*qname addr:* RX-type address or register (2) - (12). *rname addr:* RX-type address or register (2) - (12). *rname length:* decimal digit, register (2) - (12). **Default:** assembled length of rname. **Default:** *qname length* of eight.
,SYSNAME=*(sysname addr [,asid value])*	*sysname addr:* RX-type address or register (2) - (12). *asid value:* symbol, decimal digit, or register (2) - (12). **Notes:** *rname addr* can be provided only when *qname addr* is used. *rname length* must be coded if a register is specified for *rname addr*. An *asid value* can be coded only when the *sysname addr* is used.
,QUIT=YES ,QUIT=NO	**Default:** QUIT=NO **Note:** QUIT=YES is mutually exclusive with all parameters but TOKEN.
,REQCNT=*value* ,OWNERCT=*value*,WAITCNT=*value* ,OWNERCT=*value* ,WAITCNT=*value*	*value:* decimal digit or register (2) - (12). **Default:** REQCNT=0
,TOKEN=*addr*	*addr:* RX-type address or register (2) - (12).

The parameters are explained as follows:

AREA= *(area addr, area size)*
specifies the location and size of the area where information extracted from the global resource serialization resource queues is to be placed. The minimum size is the amount needed to describe a single resource, approximately 296 bytes, which is the length of the fixed portions of the RIB and the maximum size rname rounded up to a fullword value.

,REQLIM=*value*
,REQLIM=MAX

specifies the maximum number of owners and waiters to be returned for each resource, which can be any value between 0 and 2^{15}-1. MAX specifies 2^{15}-1.

,SCOPE=ALL
,SCOPE=STEP
,SCOPE=SYSTEM
,SCOPE=SYSTEMS
,SCOPE=LOCAL
,SCOPE=GLOBAL

specifies that you want information only for resources having the indicated scope. STEP, SYSTEM, or SYSTEMS is the scope specified on the resource request. If you specify SCOPE=ALL (meaning STEP, SYSTEM, and SYSTEMS), the system returns information for all resources the system recognizes that have the specified RESNAME, RESERVE, or SYSNAME characteristics. If you specify SCOPE=LOCAL, information is returned about this system's resources that are not being shared with other systems in the ring. If you specify SCOPE=GLOBAL, information is returned about resources that are being shared with other systems in the ring. Remember that entries in the resource name lists can cause the scope to change.

,RESERVE=YES
,RESERVE=NO
,RESNAME=*(qname addr[,rname addr,rname length],[GENERIC|SPECIFIC], qname length)*
,SYSNAME=*(sysname addr [,asid value])*

For most requests, RESERVE=YES specifies that information is to be returned for resources requested with the RESERVE macro instruction. If a RESERVE macro instruction is issued for a device that is not shared, global resource serialization treats the RESERVE request as an ENQ and the GQSCAN macro instruction does not return information for the resource request when RESERVE=YES.

RESERVE=NO specifies that information is to be returned for resources requested with the ENQ macro instruction.

RESNAME (with *rname)* indicates the name of one resource.

The *qname addr* specifies the virtual storage address of the 8-character major name of the requested resource.

The *rname addr* specifies the virtual storage address of a 1 to 255-byte minor name used in conjunction with the major name to represent a single resource. Information returned is for a single resource unless you specify SCOPE=ALL, in which case it could be for three resources (STEP, SYSTEM, and SYSTEMS) or SCOPE=LOCAL in which case it could be for two resources (STEP and SYSTEM) if there is a matching name in each of these categories. If the name specified by *rname* is defined by an EQU assembler instruction, the *rname length* must be specified.

The *rname length* specifies the length of the minor name. If you use the register form, the low-order (rightmost) byte contains the length. The length must match the *rname length* specified on ENQ or RESERVE.

GENERIC specifies that the rname of the requested resource must match but only for the length specified. For example, an ENQ for SYS1.PROCLIB would match the GQSCAN rname specified as SYS1 for an rname length of 4.

SPECIFIC specifies that the rname of the requested resource must exactly match the GQSCAN rname.

Note: GENERIC and SPECIFIC are mutually exclusive.

The qname length specifies the length of the qname in the resource name that must match the GQSCAN name.

SYSNAME specifies that information is to be returned for resources requested by tasks running on an MVS system whose system name matches the one specified by SYSNAME, where *sysname addr* is the address of an 8-byte field that contains the system name, and asid value specifies a 4-byte address space identifier, right justified. Information returned includes only those resources whose *sysname addr* and *asid value* match the ones specified. SYSNAME=0 or SYSNAME=(0,asid value), specifies that the system name is that of the system on which GQSCAN is issued.

,QUIT=YES
,QUIT=NO
indicates whether or not you want to terminate the current global resource serialization queue scan. If QUIT=YES is specified with TOKEN, GQSCAN processing terminates the current GRS queue scan and releases the storage allocated to accumulate the information specified in the token.

,REQCNT=*value*
,OWNERCT=*value*,**WAITCNT=***value*
,OWNERCT=*value*
,WAITCNT=*value*
specifies that you only want information about resources that fall into the following categories:

- The total number of requestors (that is, owners plus waiters) is greater than or equal to REQCNT.
- The total number of owners is greater than or equal to OWNERCT.
- The total number of waiters is greater than or equal to WAITCNT.

If you do not specify REQCNT, you can specify both OWNERCT and WAITCNT. If you specify REQCNT, you cannot specify either OWNERCT or WAITCNT.

,TOKEN=*addr*
specifies the address of a fullword of storage that the GQSCAN service routine can use in subsequent invocation to provide you with any remaining information. If the token is zero, the scan starts at the beginning of the resource queue. You must zero the token each time you want the scan to start over. If the token is not zero, the scan resumes at the point indicated by the token.

When GQSCAN returns control, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
0	Queue scan processing is complete. Data is now in the area you specified. On a resumed GQSCAN, the code signifies that there are no more resources to match your request.
4	Queue scan processing is complete. No resources matched your request.
8	The area you specified was filled before queue scan processing completed. If you specified TOKEN, process the information in the area and issue GQSCAN again specifying the TOKEN returned to you. If you did not specify TOKEN, you must begin again and either specify a larger area or specify a TOKEN.
C	Queue scan encountered an abnormal situation while processing. The information in your area is not meaningful. The values in registers 0 and 1 are also meaningless.
10	An invalid SYSNAME was specified as input to queue scan. The information in your area is not meaningful.
14	The area you specified was filled before queue scan processing completed. Your request specified TOKEN=, but you have too many outstanding ENQ or RESERVE and GQSCAN requests. The information in your area is valid but incomplete. The scan cannot be resumed.

GQSCAN (List Form)

The list form of the GQSCAN macro instruction is used to construct a non-executable parameter list. This parameter list, or a copy of it for reentrant programs, can be referred to by the execute form of the GQSCAN macro instruction.

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

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

AREA=*(area addr, area size)*	*area addr:* A-type address. *area size:* symbol, decimal digit. **Notes:** **1.** This parameter cannot be specified with QUIT=YES. **2.** AREA is required on either the list or the execute form of the macro instruction.
,REQLIM=*value* ,REQLIM=MAX	*value:* symbol, decimal digit or the word MAX. **Default:** REQLIM=MAX
,SCOPE=ALL ,SCOPE=STEP ,SCOPE=SYSTEM ,SCOPE=SYSTEMS	**Default:** SCOPE=STEP
,RESERVE=YES ,RESERVE=NO	**Default:** All resources requested with RESERVE and all resources requested with ENQ.
,RESNAME=*(qname addr [,rname addr, rname length], [GENERIC|SPECIFIC], qname length)*	*qname addr:* A-type address. *rname addr:* A-type address. *rname length:* decimal digit. **Default:** assembled length of rname. **Default:** *qname length* of eight.
,SYSNAME=*(sysname addr[,asid value])*	*sysname addr:* A-type address. *asid value:* symbol, decimal digit. **Notes:** *rname addr* can be provided only when *qname addr* is used. *rname length* must be provided if a register is specified for *rname addr*. An *asid value* can be coded only when the *sysname addr* is used.
,QUIT=YES ,QUIT=NO	**Default:** QUIT=NO **Note:** Only TOKEN and MF=L can be specified with QUIT=YES.
,REQCNT=*value*	*value:* decimal digit. **Default:** REQCNT=0
,OWNERCT=*value* ,WAITCNT=*value*	
,TOKEN=*addr*	*addr:* RX-type address.
,MF=L	

The parameters are explained under the standard form of the GQSCAN macro instruction with the following exception:

,MF = L
specifies the list form of the GQSCAN macro instruction.

GQSCAN (Execute Form)

The execute form of the GQSCAN macro instruction can refer to and modify a remote parameter list built by the list form of the macro. There are no defaults for any of the parameters in the execute form of the macro instruction.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede GQSCAN.
GQSCAN	
b	One or more blanks must follow GQSCAN.
AREA=*(area addr,area size)*	*area addr:* RX-type address or register (2) - (12). *area size:* symbol, decimal digit, or register (2) - (12). **Notes:** **1.** AREA cannot be specified with QUIT=YES. **2.** AREA is required on either the list or the execute form of the macro instruction.
,REQLIM=*value* ,REQLIM=MAX	*value:* symbol, decimal digit, register (2) - (12), or the word MAX.
,SCOPE=ALL ,SCOPE=STEP ,SCOPE=SYSTEM ,SCOPE=SYSTEMS ,SCOPE=LOCAL ,SCOPE=GLOBAL	**Note:** SCOPE=LOCAL and SCOPE=GLOBAL cannot be coded on the list form of this macro.
,RESERVE=YES ,RESERVE=NO	**Default:** All resources requested with RESERVE and all resources requested with ENQ.
,RESNAME=*(qname addr[,rname addr,rname length], [GENERIC|SPECIFIC], qname length)*	*qname addr:* RX-type address or register (2) - (12). *rname addr:* RX-type address or register (2) - (12). *rname length:* decimal digit, register (2) - (12). **Default:** assembled length of rname.
,SYSNAME=*(sysname addr[,asid value])*	*sysname addr:* RX-type address or register (2) - (12). *asid value:* symbol, decimal digit, or register (2) - (12). **Note:** *rname addr* can be provided only when *qname addr* is used. *rname length* must be provided if a register is specified for *rname addr*. An *asid value* can be coded only when the *sysname addr* is used.
,QUIT=YES ,QUIT=NO	**Default:** QUIT=NO **Note:** Only TOKEN and MF=(E, *parm list addr*) can be specified with QUIT=YES.
,REQCNT=*value*	*value:* decimal digit or register (2) - (12).
,OWNERCT=*value* ,WAITCNT=*value*	
,TOKEN=*addr*	*addr:* RX-type address of register (2) - (12).
,MF=*(E,parm list addr)*	*parm list addr:* RX-type address or register (2) - (12).

The parameters are explained under the standard form of the GQSCAN macro instruction with the following exception:

,MF = (E,*parm list addr)*

specifies the execute form of the GQSCAN macro instruction. This form uses a remote parameter list defined by parm list addr.

IEFQMREQ - Invoke SWA Manager in Move Mode

This macro is used to invoke the Move SWA manager in move mode. The IEFQMREQ macro instruction, which has no parameters, is written as follows:

name	*name:*
b	One or more blanks must precede IEFQMREQ.
IEFQMREQ	
b	One or more blanks must follow IEFQMREQ.

INTSECT - Intersect With the Dispatcher

This macro can be assembled compatibly between MVS/XA and MVS/370 through the use of the SPLEVEL macro instruction. Default processing will result in an expansion of the macro that operates only with MVS/XA. See the topic "Selecting the Macro Level" for additional information. If you are executing in 31-bit addressing mode, you must use the MVS/XA version of this macro instruction.

The INTSECT macro instruction is used to serialize processing with the dispatcher (specifically, when altering the dispatching queues or TCB/ASCB dispatchability). The only routines that may use the INTSECT macro are those in supervisor state, key 0.

There are two levels of intersect, local and global. The LOCAL lock must be held before requesting the local intersect. The dispatcher lock must be held before requesting the global intersect. After the operation that required serialization has completed, the intersect must be released by issuing the INTSECT macro instruction with the RESET option before freeing the appropriate lock. Programs executing in cross memory mode can use INTSECT with the TYPE=GLOBAL option.

Note: CML (cross memory local) lock means the local lock of an address space other than the home address space. LOCAL lock means the local lock of the home address space. When written in lower case, local lock means any local-level lock, either the LOCAL or a CML lock.

The INTSECT macro instruction is written as follows:

name	*name:*
b	One or more blanks must precede INTSECT.
INTSECT	
b	One or more blanks must follow INTSECT.

SET RESET	
,TYPE=GLOBAL ,TYPE=LOCAL	
,ID=*symbol*	
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

SET
RESET
specifies whether to obtain or release the intersect.

,TYPE=GLOBAL
,TYPE=LOCAL
specifies the level of intersect to be used.

,ID=*symbol*
specifies an identifier that indicates the function name of the intersecting routine.

Note: Specific ids are defined only for the control program. Other users should omit this keyword. Routines omitting the ID keyword should release the INTSECT before calling any other routine.

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

Example 1

Operation: The LOCAL lock has already been obtained. Request to hold the local intersect.

```
INTSECT SET,TYPE=LOCAL
```

IOSDDT - Device Descriptor Table Build Macro

The IOSDDT macro must be included in the unit information module (UIM) that an installation provides for any device that the MVS configuration program (MVSCP) does not support. The IOSDDT macro builds a device descriptor table (DDT). The DDT, which must reside in SYS1.NUCLEUS, is the vector table to the device dependent exits for a device. The IOSDDT macro is located in SYS1.AMODGEN. See *SPL: System Modifications* for a complete description of coding a UIM.

A device descriptor table (DDT) is a vector table that IOS uses to locate the device support routines. The system requires one of these tables for each device in the I/O configuration, although similar devices may share the same DDT. When conditions arise during I/O operations for which specific device dependent processing is required, IOS gives control to the exit routines through the vector entries in the DDT.

To build the DDT, you use the IOSDDT macro. With this macro, you specify the module names of the DDT exit routines for the devices supported by that DDT. These exit routines perform the processing for various system functions that occur when the system performs I/O operations. The parameters of the IOSDDT macro allow you to specify the following kinds of routines, which receive control from IOS when the appropriate condition arises:

- The start I/O exit routine
- The trap exit routine
- The translate CCW table
- The ERP message routine
- The DDR exit routine
- The unsolicited interrupt exit routine
- The sense exit routine
- The end of sense exit routine
- The MIH exit routine
- The device initialization exit routine
- The channel program scan exit routine
- The subsystem ID

The information in the DDT is created from the parameters of the IOSDDT macro. The label that you specify on the IOSDDT macro is required because it is used as the CSECT name for the DDT being generated. When the system is IPLed, the DDT for each device in the I/O configuration becomes part of the nucleus. Each use of the IOSDDT macro generates one DDT.

The IOSDDT macro instruction is written as follows:

name	*name:*
b	One or more blanks must precede IOSDDT.
IOSDDT	
b	One or more blanks must follow IOSDDT.

SIOEXIT=*epname*	*entry point name*
,TRPEXIT=*epname*	*entry point name*
,TCCWTAB=*epname*	*entry point name*
,ERPEXIT=*(epname,type)*	*entry point name*
,DDREXIT=*(epname,type)*	*entry point name*
,UNSEXIT=*epname*	*entry point name*
,SNSEXIT=*epname*	*entry point name*
,EOSEXIT=*epname*	*entry point name*
,MIHEXIT=*epname*	*entry point name*
,DSEXIT=*epname*	*entry point name*
,CPSEXIT=*epname*	*entry point name*
,SSYSID=*ssname*	*subsystem name*

The parameters are explained as follows:

name
specifies name of the DDT. IOSDDT uses this name on the CSECT statement that it generates for the DDT. The name parameter is required.

SIOEXIT=*epname*
specifies the name of the start I/O exit entry point.

TRPEXIT=*epname*
specifies the name of the trap exit entry point.

TCCWTAB=*epname*
specifies the name of the translate CCW table entry point.

ERPEXIT = *(epname,type)*
specifies the name of the ERP message entry point. Type describes whether the entry point name is to be treated as an entry point name address or a module name. Type can be specifed as A for address or N for EBCDIC name. If A is specified, the module is loaded into the nucleus region from SYS1.NUCLEUS. . If N is specified, the module is loaded into the LPA from the LINK LIST concatenation. If neither is specified, N is the default.

DDREXIT = *(epname,type)*
specifies the name of the DDR exit entry point Type describes whether the entry point name is to be treated as an entry point name address or a module name. Type can be specifed as A for address or N for EBCDIC name. If A is specified, the module is loaded into the nucleus region from SYS1.NUCLEUS. . If N is specified, the module is loaded into the LPA from the LINK LIST concatenation. If neither is specified, N is the default.

UNSEXIT = *epname*
specifies the name of the unsolicited interrupt exit entry point.

SNSEXIT = *epname*
specifies the name of the sense exit entry point.

EOSEXIT = *epname*
specifies the name of the end of sense exit entry point.

MIHEXIT = *epname*
specifies the name of the MIH exit entry point.

DSEXIT = *epname*
specifies the name of the device service exit entry point.

CPSEXIT = *epname*
specifies the name of the channel program scan exit entry point.

SSYSID = *ssname*
specifies the name of the subsystem ID, which can be one to four characters.

Note: When both ERPEXIT and DDREXIT are specified as EBCDIC module names, IOSDDT verifies that both specified module names have the same 4-character prefix. If the prefixes are not the same, IOSDDT issues an MNOTE and not does generate a DDT.

IOSDMLT - Module Lists Table Macro

The IOSDMLT macro must be included in the unit information module (UIM) that an installation provides for any device that the MVS configuration program (MVSCP) does not support. The IOSDMLT macro builds a module list table (MLT). See *SPL: System Modifications* for a complete description of coding a UIM.

The Module Lists Table (MLT) must reside in SYS1.NUCLEUS. It identifies the nucleus and LPA modules required to support the device you are defining, and that need to be loaded during the IPL process. For example, the MLT for an unsupported printer would designate all the modules that must be loaded into the nucleus and the LPA to support that printer. Note that the MLT must list all the nucleus and LPA device support modules for the device regardless of whether the modules are provided by you or by IBM.

To build a module lists table, use the IOSDMLT macro. Each IOSDMLT macro that you code creates an MLT CSECT. The label specified on the IOSDMLT macro, which is required, is used as the CSECT name. As parameters of the IOSDMLT macro, you specify a set of nucleus-resident module names and a set of LPA-resident module names. Each use of the IOSDMLT macro generates one MLT, which resides in a separate module. The IOSDMLT macro resides in SYS1.AMODGEN.

The IOSDMLT macro instruction is written as follows:

name	*name:*
b	One or more blanks must precede IOSDMLT.
IOSDMLT	
b	One or more blanks must follow IOSDMLT.

NUCL=*(nucid<,nucid>...)*	*nucid:* name of nucleus module
,LPAL=*(lpaid<,lpaid>...)*	*lpaid:* name of LPA module

The parameters are explained as follows:

name
specifies the name of the MLT. IOSDMLT uses this name on the CSECT statement that it generates for the MLT. The name parameter is required. Note that IOSDMLT generates an END statement at the end of its expansion.

NUCL = *(nucid<,nucid>...)*
specifies the names of the nucleus modules that are to be loaded from SYS1.NUCLEUS into the nucleus region if the device associated with this MLT is defined in the I/O configuration.

,LPAL = *(lpaid<,lpaid>...)*
specifies the names of the LPA modules to be included in the module list table CSECT.

IOSINFO - Obtain Information From the Input/Output Supervisor (IOS)

The IOSINFO macro instruction obtains the subchannel number for a specified unit control block (UCB) from the input/output supervisor (IOS). The macro returns the subsystem identification word (SID), which identifies the subchannel number of the UCB, in a user-specified location. The SID is a fullword value whose first halfword contains X'0001' and ending halfword contains the subchannel number.

A subchannel is associated with a UCB at NIP time. If the subchannel and the UCB become disassociated during system operation, the subchannel number in the SID might not be valid. If the UCB is disassociated from the subchannel at the time that the IOSINFO service routine is invoked, IOSINFO can detect the situation and notify the user via a return code. If the UCB is disassociated from the subchannel after the service routine is invoked, IOSINFO can give no notification of this to the caller.

The issuer of IOSINFO must be executing:

- In 31-bit addressing mode
- In either task mode or SRB mode
- Locked or unlocked

Additionally, the issuing program uses the CVT and PSA control blocks. All addresses must be 31-bit addresses.

Before entry to this macro, register 13 must contain the address of a standard 18-word save area.

The IOSINFO macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSINFO.
IOSINFO	
b	One or more blanks must follow IOSINFO.
FUNCTN=SUBCHNO	
,UCB=*ucb addr*	*ucb addr:* A-type address or register (0) - (15).
,OUTPUT=*output addr*	*output addr:* A-type address or register (0) - (14).
,RTNCDE=*retcde addr*	*retcde addr:* A-type address or register (0) - (15).

The parameters are explained as follows:

FUNCTN=SUBCHNO
specifies that a subchannel number is to be obtained.

,UCB=*ucb addr*
specifies the address of a fullword on a fullword boundary containing the address of a unit control block (UCB).

,OUTPUT=*output addr*
specifies the address of a fullword on a fullword boundary that will contain the subsystem identification word (SID) upon completion.

The SID is a fullword value that identifies the subchannel. The first halfword is X'0001', and the last halfword contains the subchannel number.

The output address must reside in 31-bit addressable storage.

,RETCDE=*retcde addr*
specifies the address of a fullword on a fullword boundary that will contain the return code upon completion.

The return code address must reside in 31-bit addressable storage.

After completion, the contents of the registers are as follows:

- Register 0 is unpredictable.
- Register 1 (unless the return code is 4) contains the SID.
- Registers 2-13 are preserved.
- Register 14 is unpredictable.
- Register 15 contains a return code.

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

Hexadecimal Code	Meaning
00	The address specified on the OUTPUT parameter contains the SID.*
04	The UCB was disassociated from the subchannel at the time of the IOSINFO service routine invocation.

* In some cases, the subchannel number in the SID might not be valid. Any disassociation of the UCB and the subchannel means the subchannel number in the SID is not valid. If the UCB is disassociated from the subchannel after the IOSINFO service routine invocation, no notification can be given.

Example 1

Operation: Obtain the subchannel number for a UCB whose address is in register 1. Specify the SID output to be placed in register 2 and the return code to be placed in register 3.

```
IOSINFO  FUNCTN=SUBCHNO,UCB=(1),OUTPUT=(2),RTNCODE=(3)
```

Example 2

Operation: Obtain the subchannel number for a UCB whose address is in location ADDR. Specify the SID output to be placed in location ADDX and the return code to be placed in register 3.

```
IOSINFO  FUNCTN=SUBCHNO,UCB=ADDR,OUTPUT=ADDX,RTNCODE=(3)
```

Example 3

Operation: Obtain the subchannel number for a UCB whose address is in register 2. Specify the SID output to be placed in register 3 and the return code to be placed in location ADDR.

```
IOSINFO  FUNCTN=SUBCHNO,UCB=(2),OUTPUT=(3),RTNCODE=ADDR
```

IOSLOOK - Locate Unit Control Block

The IOSLOOK macro instruction locates the unit control block (UCB) associated with a device address. To use IOSLOOK, you must be executing in supervisor state. Register 13 must point to a 16-word save area where the macro instruction stores registers 0 through 15 at offset 0. You must also include a DSECT for both the CVT (using the CVT mapping macro) and the IOCOM (using the IECDIOCM mapping macro).

The IOSLOOK macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSLOOK.
IOSLOOK	
b	One or more blanks must follow IOSLOOK.
DEV=*(reg)*	*reg:* Register (0) - (12), (14), (15). **Default:** DEV=(6).
,UCB=*(reg)*	*reg:* Register (0) - (12). **Default:** UCB=(7).

The parameters are explained as follows:

DEV=*(reg)*
specifies a general purpose register, symbolic or absolute, that contains the hexadecimal device number, right justified. If this parameter is omitted, register 6 is assumed.

,UCB=*(reg)*
specifies a general purpose register, symbolic or absolute, that will be used to return the address of the UCB common segment. If this parameter is omitted, register 7 is assumed.

If the UCB address cannot be found, then the contents of this register are unpredictable.

Note: The UCB must reside in 24-bit addressable storage.

When control returns, register 15 contains one of the following return codes.

Hexadecimal Code	Meaning
00	UCB address was found
04	Device number was invalid or no UCB exists.

Example 1

Operation: Find the UCB address for device 250. Register 2 contains the value X'00000250'. The UCB address is to be returned in register 5 and UCBPTR is equated to 5.

```
IOSLOOK DEV=(2),UCB=(UCBPTR)
```

LOAD - Bring a Load Module into Virtual Storage

The LOAD macro instruction is used to bring the load module containing the specified entry name into virtual storage, if a usable copy is not available in virtual storage. Load services places the load module in storage above or below the 16 megabytes line depending on the RMODE of the module, which is specified in the directory entry for the module.

The responsibility count for the load module is increased by one. On output, the high-order byte of register 1 contains the authorization code of the loaded module and the low-order three bytes contain the module's length in doublewords. Control is *not* passed to the load module; instead, the virtual storage address and the addressing mode of the designated entry point is returned in register 0. The load module remains in virtual storage until the responsibility count is reduced to 0 through task terminations or until the effects of all outstanding LOAD requests for the module have been canceled (using the DELETE macro instruction described in *Supervisor Services and Macro Instructions*), *and* there is no other requirement for the module.

Load sets the high-order bit of the entry point address in register 0 to indicate the module's AMODE, which is obtained from the directory entry for the module. If the module's AMODE is 31-bit, it sets the indicator to 1; if the module's AMODE is 24-bit, it sets the indicator to 0; and if the module's AMODE is ANY, it sets the indicator to correspond to the caller's AMODE.

The GLOBAL,EOM, and ADDR parameters are restricted to authorized users (APF-authorized, in PSW key 0-7, or in supervisor state).

The entry name in the load module must be a member name or an alias in a directory of a partitioned data set or must have been specified in an IDENTIFY macro instruction. If the entry name was previously specified in an IDENTIFY macro instruction, no attempt is made to bring in an additional copy of the module. If the specified entry name cannot be located, the task is abnormally terminated.

The LOAD macro instruction is written as follows:

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

EP=*entry name* EPLOC=*entry name addr* DE=*list entry addr*	*entry name:* symbol. **entry name addr:* RX-type address or register (2) - (12); A-type *address or register (2) - (12).* **list entry addr:* RX-type address, or register (2) - (12); A-type address or register (2) - (12).
,DCB=*dcb addr*	**dcb addr:* RX-type address, or register (1) or (2) - (12); A-type address or register (2) - (12).
,ERRET=*err rtn addr*	*err rtn addr:* RX-type address, or register (2) - (12).
,LSEARCH=NO ,LSEARCH=YES	**Default:** LSEARCH=NO
,ADDR=*load addr*	*load addr:* A-type address or register (2) - (12).
,GLOBAL=YES ,GLOBAL=(YES,P) ,GLOBAL=(YES,F) ,GLOBAL=NO	**Default:** GLOBAL=NO If GLOBAL=YES is specified, the default is GLOBAL=(YES,P).
,EOM=NO ,EOM=YES	**Default:** EOM=NO **Note:** GLOBAL must be specified with EOM=YES.
,LOADPT=*addr*	*addr:* A-type address or register (2) - (12). **Note:** ADDR cannot be specified with LOADPT.
,RELATED=*value*	*value:* any valid macro keyword specification.

* If you code any of the parameters: LSEARCH, ADDR, GLOBAL, EOM, or LOADPT, you will obtain a macro-generated parameter list. Therefore, except for the error routine address, all addresses must be specified as A-type addresses or registers (2) - (12).

The parameters are explained below:

EP=*entry name*
EPLOC=*entry name addr*
DE=*list entry addr*

specifies the entry name, the address of the name, or the address of the name field in a 60-byte list entry for the entry name that was constructed using the BLDL macro instruction. If EPLOC is coded, the name must be padded to eight bytes, if necessary.

Note: The task structure must not be changed via an ATTACH or DETACH between the issuance of the BLDL and the issuance of the ATTACH for the module, or an abend 106 with a return code of 15 might result.

,DCB=*dcb addr*

specifies the address of the data control block for the partitioned data set containing the entry name described above. This parameter must indicate the same DCB used in the BLDL mentioned above.

If the DCB parameter is omitted or if DCB=0 is specified when the LOAD macro instruction is issued by the job step task, the data sets referred to by either the STEPLIB or JOBLIB DD statement are first searched for the entry name. If the entry name is not found, the link library is searched.

If the DCB parameter is omitted or if DCB=0 is specified when the LOAD macro instruction is issued by a subtask, the data sets associated with one or more data control blocks referred to by the TASKLIB operand of previous ATTACH macro instructions in the subtask chain are first searched for the entry name. If the entry name is not found, the search is continued as if the LOAD had been issued by the job step task.

Note: DCB must reside in 24-bit addressable storage.

,ERRET=*err rtn addr*

specifies the address of a routine to receive control when an error condition that would cause an abnormal termination of the task is detected. Register 1 contains the abend code that would have resulted had the task abended, and register 15 contains the reason code that is associated with the abend. The routine does not receive control when input parameter errors are detected.

,LSEARCH=NO
,LSEARCH=YES

specifies whether (YES) or not (NO) you want the library search limited to the job pack area and to the first library in the normal search sequence.

,ADDR=*load addr*

specifies that the module is to be loaded beginning at the designated address. The address must specify a doubleword boundary. Storage for the module must have been previously allocated in the requestor's key. The system does not search for the module and does not maintain a record of the module once it is loaded. If you code the ADDR parameter, you must also code the DCB parameter (not DCB=0) and you must not code GLOBAL or LOADPT.

Note: The RMODE of the load module must agree with this address. If the user specifies an address above 16 megabytes in virtual, the load module must have an RMODE of ANY.

,GLOBAL=YES
,GLOBAL=(YES,P)
,GLOBAL=(YES,F)
,GLOBAL=NO

specifies whether the module is to be loaded into the pageable common service area (CSA) (GLOBAL=(YES,P) or GLOBAL=YES), loaded into fixed CSA (GLOBAL=(YES,F)), or not loaded into CSA (GLOBAL=NO). (The module must not have been previously loaded into CSA with different attributes by the same job step, the module must also be reentrant and must reside in an APF-authorized library.) For GLOBAL=(YES,F), the module must not be marked as requiring alignment on a page boundary. If you code the GLOBAL parameter, you cannot code the ADDR parameter.

If the requested module resides in the link pack area, the LOAD request performs as though the GLOBAL parameter was omitted. The LOAD request locates the module in the link pack area, allows access to it, but does not load a copy of the desired module into the common service area.

Note: A load request with the GLOBAL option does not cause the loaded module to be implicitly known to other address spaces. The loaded module can be accessed by other address spaces, however, only the requesting task is accountable for it (and may therefore delete it).

,EOM=YES
,EOM=NO
indicates whether a module in global storage is to be deleted when the address space terminates (EOM=YES) or when the task terminates (EOM=NO). If you code EOM, you must also code GLOBAL.

,LOADPT=*addr*
specifies that the starting address at which the module was loaded is to be returned to the caller at the indicated address. If you code LOADPT, you cannot code ADDR.

,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.

Example 1

Operation: Bring a load module with entry name PGMLKRUS into virtual storage. Let the system find the module from available libraries.

```
LOAD   EP=PGMLKRUS
```

Example 2

Operation: Bring a load module with entry name PGMEOM into pageable CSA storage and return the load address at location PGMLPT.

```
LDPGM    LOAD  EP=PGMEOM,EOM=YES,LOADPT=PGMLPT,GLOBAL=(YES,P)
         .
         .
         .
PGMLPT   DS    A                      LOAD ADDRESS RETURNED HERE
```

LOAD (List Form)

The list form of the LOAD macro instruction builds a non-executable parameter list that can be referred to by the execute form of the LOAD macro.

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

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

EP=*entry name* EPLOC=*entry name addr* DE=*list entry addr*	*entry name:* symbol. *entry name addr:* A-type address. *list entry addr:* A-type address.
,DCB=*dcb addr*	*dcb addr:* A-type address.
,LSEARCH=NO ,LSEARCH=YES	**Default:** LSEARCH=NO
,ADDR=*load addr*	*load addr:* A-type address.
,GLOBAL=YES ,GLOBAL=(YES,P) ,GLOBAL=(YES,F) ,GLOBAL=NO	**Default:** GLOBAL=NO If GLOBAL=YES is specified, the default is GLOBAL=(YES,P).
,EOM=NO ,EOM=YES	**Default:** EOM=NO **Note:** GLOBAL must be specified with EOM=YES.
,LOADPT=*addr*	*addr:* A-type address. **Note:** ADDR cannot be specified with LOADPT.
,RELATED=*value*	*value:* any valid macro keyword specification.
,SF=L	

The parameters are explained under the standard form of LOAD macro instruction with the following exception:

,SF=L
specifies the list form of the LOAD macro instruction.

LOAD (Execute Form)

The execute form of the LOAD macro instruction can refer to and modify the parameter list constructed by the list form of the macro.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede LOAD.
LOAD	
b	One or more blanks must follow LOAD.
EP=*entry name* EPLOC=*entry name addr* DE=*list entry addr*	*entry name:* symbol. *entry name addr:* RX-type address or register (2) - (12). *list entry addr:* RX-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr:* RX-type address, or register (2) - (12).
,ERRET=*err rtn addr*	*err rtn addr:* RX-type address, or register (2) - (12).
,LSEARCH=NO ,LSEARCH=YES	**Default:** LSEARCH=NO
,ADDR=*load addr*	*load addr:* RX-type address or register (2) - (12). **Note:** For an RX-type address, the operand is treated as the address of a field that contains the actual load address.
,GLOBAL=YES ,GLOBAL=(YES,P) ,GLOBAL=(YES,F) ,GLOBAL=NO	**Default:** GLOBAL=NO **Note:** If GLOBAL=YES is specified, the default is GLOBAL=(YES,P).
,EOM=NO ,EOM=YES	**Default:** EOM=NO **Note:** GLOBAL must also be specified with EOM=YES.
,LOADPT=*addr*	*addr:* RX-type address or register (2) - (12). **Note:** ADDR cannot be specified with LOADPT.
,RELATED=*value*	*value:* any valid macro keyword specification.
,SF=*(E,list addr)*	*list addr:* RX-type address or register (2) - (12) or (15).

The parameters are explained under the standard form of LOAD macro instruction with the following exception:

,SF=(E,*list addr***)**
specifies the execute form of the LOAD macro instruction.

LOCASCB - Locate ASCB

The LOCASCB macro instruction is used to locate the ASCB address associated with a specified ASID.

The LOCASCB macro instruction uses registers 0, 1, 14, and 15.

If the caller is concerned that the ASCB might terminate while being referenced, the caller should provide some serialization to prevent ASCB termination by holding either the CMS lock or the dispatcher lock.

Programs executing in cross memory mode can invoke the LOCASCB macro instruction.

The LOCASCB macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede LOCASCB.
LOCASCB	
b	One or more blanks must follow LOCASCB.
ASID=*asid addr*	*asid addr:* RX-type address or register (0) - (15).

The parameter is explained as follows:

ASID=*asid addr*
specifies the RX-type address of a halfword that contains the ASID whose ASCB is to be located or the register that contains the ASID in bits 16-31. (Bits 0-15 of the register are ignored.) If the caller specifies (1), the ASID need not be copied into register 1 by the macro expansion.

When LOCASCB returns control, register 1 contains the results of the locate operation as follows:

- If register 1 is positive, it contains the ASCB address.
- If register 1 is negative or zero, the specified ASID is invalid.

LXFRE - Free a Linkage Index

The LXFRE macro instruction frees one or more linkage indexes. You cannot free a linkage index that was reserved with the SYSTEM option. (See the LXRES macro instruction). Before issuing the LXFRE macro instruction, disconnect all entry tables associated with the linkage index, unless you specify FORCE=YES. If you do not disconnect the entry tables and do not specify FORCE=YES, linkage indexes are not freed and the routine is abnormally terminated.

The requestor must be in supervisor state or PKM 0-7 executing in primary mode enabled and unlocked. Register 13 must point to a standard register save area that must be addressable in primary mode. The parameter list passed to this macro instruction must also be addressable in primary mode when the macro instruction is issued.

Registers 2-14 are preserved. Register 2, which is modified by the macro after the registers are saved, should not be used as the base register. Register 15 contains the return code. The contents of registers 0 and 1 are unpredictable.

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

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

LXLIST=*list addr*	*list addr:* RX-type address or register (0) - (12).
,FORCE=NO ,FORCE=YES	**Default:** FORCE=NO
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

LXLIST=*list addr*
specifies the address of a variable length list of fullword entries. The first word in the list must contain the number (1 to 32) of linkage indexes to be freed. Each entry following the first must contain a linkage index value specified in the form returned by the LXRES macro instruction.

,FORCE=NO
,FORCE=YES

specifies whether (YES) or not (NO) the linkage index is to be freed even if entry tables are currently connected to it. Any connected entry tables are disconnected before the linkage index is freed. FORCE=NO is the default.

,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 can be any valid coding values.

When LXFRE returns control, register 15 contains contains one of the following return codes:

Hexadecimal Code	Meaning
0	The specified linkage indexes were freed. No entry tables were connected.
4	The specified linkage indexes were freed. Entry tables were connected, but FORCE was specified and was successfully executed.
8	Some of the specified linkage indexes were freed. Entry tables were connected. FORCE was specified but one or more of the necessary disconnects failed. No action by the issuer of LXFRE is required in this situation.

LXFRE (List Form)

The list form of the LXFRE macro instruction is used to construct a non-executable parameter list. The execute form of the LXFRE macro instruction can refer to or modify the parameter list.

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

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

LXLIST=*list addr*	*list addr:* A-type address.
,FORCE=NO ,FORCE=YES	**Default:** FORCE=NO
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

The parameters are explained under the standard form of the LXFRE macro instruction with the following exception:

,MF=L
specifies the list form of the LXFRE macro instruction.

LXFRE (Execute Form)

The execute form of the LXFRE macro instruction can refer to and modify a remote parameter list created by the list form of the macro.

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

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

LXLIST=*list addr*	*list addr:* RX-type address or register (0) - (12).
,FORCE=NO ,FORCE=YES	**Default:** FORCE=NO
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=*(E,cntl addr)*	*cntl addr:* RX-type address or register (0) - (12).

The parameters are explained under the standard form of the LXFRE macro instruction with the following exception:

,MF=(E,*cntl addr)*
specifies the execute form of the LXFRE macro instruction. This form uses a remote parameter list.

LXRES - Reserve a Linkage Index

The LXRES macro instruction reserves one or more linkage indexes for the caller's use. The reserved linkage indexes are owned by the cross memory resource ownership task of the current home address space. The linkage index reservation applies across all linkage tables in the system and remains in effect until one of the following happens:

- An LXFRE macro instruction explicitly frees a reserved linkage index.
- The cross memory resource ownership task terminates.
- The operator re-IPLs the system.

The requestor must be in supervisor state or PKM 0-7 executing in primary mode enabled and unlocked. Register 13 must point to a standard register save area that must be addressable in primary mode. The parameter list passed to the LXRES macro instruction must also be addressable in primary mode at the time the macro instruction is issued.

On return registers 3-14 are preserved, register 15 contains the return code, and the contents of registers 0 and 1 are unpredictable. Register 2, because it is modified by the macro after the registers are saved, should not be used as the base register.

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

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

LXLIST=*list addr*	*list addr:* RX-type address or register (0) - (12).
,SYSTEM=NO ,SYSTEM=YES	**Default:** SYSTEM=NO
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

LXLIST = *list addr*
specifies the address of a variable-length list of fullword entries. The first fullword in the list must contain the number (1 to 32) of linkage index values to be returned. The list must be long enough to contain the requested number of values. The linkage index values are returned in the list entries in the proper position for ORing with the entry index to form a PC number.

,SYSTEM = NO
,SYSTEM = YES
specifies whether (YES) or not (NO) the linkage indexes are being reserved for system connections. If YES is specified, a subsequent ETCON macro instruction specifying the linkage index causes all address spaces to be connected to the entry table.

,RELATED = *value*
specifies information used to self-document macro instructions by "relating" functions or services to corresponding services performed elsewhere. The format and contents of the information specified can be any valid coding values.

On return, register 15 contains the following return code:

Hexadecimal Code	Meaning
0	The specified linkage indexes were successfully reserved.

LXRES (List Form)

The list form of the LXRES macro instruction is used to construct a non-executable parameter list. The execute form of the macro instruction can then refer to this list or a copy of it for reentrant programs.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede LXRES.
LXRES	
b	One or more blanks must follow LXRES.
LXLIST = *list addr*	*list addr:* A-type address.
,SYSTEM = NO ,SYSTEM = YES	**Default:** SYSTEM = NO
,RELATED = *value*	*value:* any valid macro keyword specification.
,MF = L	

The parameters are explained under the standard form of the LXRES macro instruction with the following exception:

,MF = L
specifies the list form of the LXRES macro instruction.

LXRES (Execute Form)

The execute form of the LXRES macro instruction can refer to and modify a remote parameter list constructed by the list form of the macro instruction.

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

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

LXLIST = *list addr*	*list addr:* RX-type address or register (0) - (12).
,SYSTEM = NO ,SYSTEM = YES	**Default:** SYSTEM = NO
,RELATED = *value*	*value:* any valid macro keyword specification.
,MF = *(E,cntl addr)*	*cntl addr:* RX-type address or register (0) - (12).

The parameters are explained as under the standard form of the LXRES macro instruction with the following exception:

,MF = (E,*cntl addr)*
specifies the execute form of the LXRES macro instruction and *cntl addr* is the name or address of the list form of the macro.

MGCR - Internal START or REPLY Command

The MGCR macro instruction can be used to start a program or subsystem from within your program and to pass 31 bits of information to the started program in the form of a token. The MGCR macro instruction can also be used to issue a reply to a WTOR macro instruction.

The issuer must be in supervisor state, PSW key 0-7.

The MGCR macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede MGCR.
MGCR	
b	One or more blanks must follow MGCR.
command-buffer-address	*command-buffer-address:* RX-type address or register (1) or (2) - (12).

The parameters are explained as follows:

command-buffer-address
specifies the address of a command buffer that contains the following information.

Name	Length	Contents
flags1	1 byte	If bit 0 is one, then flags2 must contain meaningful information. Bits 1-7 must be zero.
length	1 byte	Length of the buffer up to but not including the 4-byte token field.
flags2	2 bytes	X'0800' - token is present. X'0000' - token is not present.
text	up to 126 bytes	Command, operands, and optional comments as follows: command operands comments
token	31 bits right-justified[3]	An optional field containing any desired information, such as an identifier that indicates the issuing program.

Notes:

1. *Register 0 must contain zero.*
2. *The command buffer must be located in 24-bit addressable storage.*
3. *A token is meaningful only with the START command.*

Register 15 contains one of the following return codes as the result of a START command. No return codes result from the REPLY command.

Hexadecimal Code	Meaning
00	Start command processed successfully. Register 0 contains the right justified ASID of the started address space.
08	Start command failed.

Example 1

Operation: Issue an internal start command for the catalogued procedure labeled PROG.

```
        SR      0,0
        MGCR    INPUT
        .
        .
INPUT   DC      X'80'
        DC      AL1 (TOKEN-INPUT)
        DC      X'0800'
        DC      C'S PROG'
TOKEN   DC      AL4(DATA)
```

For further examples of the internal REPLY command, refer to *User Exits.*

MODESET - Change System Status

If you are executing in 31-bit addressing mode, you must use the MVS/XA version of this macro instruction.

The MODESET macro instruction is used to change system status by altering the PSW key and/or PSW problem state indicator. It causes a supervisor routine (IEAVMODE) to alter the RB old program status word (RBOPSW) so that the desired PSW is 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 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 *Principles of Operation*) and can execute in supervisor or problem program state. If a problem state caller's key is marked as authorized in the PSW-key mask in control register 3, the inline form can execute in problem state. The inline form can be used by programs executing in cross memory mode. If the key you specify is TCB, RBT1, or RBT234, you must also ensure that current addressability is to the home address space.

The form that generates an SVC is executable by users in supervisor state, under PSW key 0-7, or APF-authorized. The SVC form cannot be used in cross memory mode.

The macro instruction does not generate any return codes.

Inline Code Generation

The standard form of the MODESET macro instruction that generates inline code is written as follows:

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

EXTKEY=*key* KEYADDR=*new key addr* KEYREG=*new key reg*	*key:* one of the following: SCHED SRM ZERO KEY2 JES SUPR TCB KEY3 RSM DATAMGT RBT1 KEY4 VSM TCAM RBT234 KEY7 *new key addr:* A-type address or register (2). **Notes:** 1. WORKREG is required if the following are specified: EXTKEY=TCB EXTKEY=RBT234 EXTKEY=RBT1 KEYADDR=A-type address 2. The WORKREG parameter should be register 1-15 if one of these four parameters is specified because WORKREG is used as a base register on the SPKA instruction. WORKREG=0 sets the PSW key to zero. *new key reg:* register 1-15 without parentheses; may be symbolic.
,SAVEKEY=*old key addr*	*old key addr:* A-type address or register (2). **Notes:** 1. If KEYADDR=(2) is specified above, then SAVEKEY=(2) cannot be specified. 2. The WORKREG parameter is required if SAVEKEY=A-type address is specified. 3. If WORKREG and SAVEKEY are specified with KEYREG, the KEYREG register should be different from the WORKREG register. Also, if SAVEKEY is specified with KEYREG, the KEYREG register should not be register 2.
,WORKREG=*work reg*	*work reg:* decimal digits 0-15 without parentheses.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

EXTKEY=*key*
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 active RB preceding SVRB of type 2, 3, or 4 SVC routine issuing MODESET.

KEY2 - Key of 2 is to be set.

KEY3 - Key of 3 is to be set.

KEY4 - Key of 4 is to be set.

KEY7 - Key of 7 is to be set.

KEYADDR=*new key addr*
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. If TCB, RBT1 or RBT234 is specified as the key address, the TCB mapping macro IKJTCB is required. The user is expected to establish addressability to the TCB with a USING statement.

KEYREG=*new key reg*
specifies a register that contains a key value in bit positions 24-27.

,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=*work 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.

SVC Generation

The standard form of the MODESET macro instruction that generates an SVC is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede MODESET.
MODESET	
b	One or more blanks must follow MODESET.
KEY=ZERO KEY=NZERO	**Note:** KEY is required if MODE is not specified.
,MODE=PROB ,MODE=SUP	**Note:** MODE is required if KEY is not specified.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

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 problem state indicator (bit 15) is to be either turned on (PROB) or turned off (SUP). If the MODESET operation completes with a problem state PSW, the PSW-key mask in control register 3 is changed to authorize only the key specified by the problem state PSW.

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
```

Example 3

Operation: Save the current key at location KEY and set the key to the value contained in bits 24-27 of register 3.

```
MODESET KEYREG=REG3,SAVEKEY=KEY,WORKREG=4
```

MODESET (List Form)

The list form of the MODESET macro instruction that generates an SVC is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede MODESET.
MODESET	
b	One or more blanks must follow MODESET.
KEY=ZERO KEY=NZERO	**Note:** KEY is required if MODE is not specified.
,MODE=PROB ,MODE=SUP	**Note:** MODE is required if KEY is not specified.
,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 exception:

,MF=L
specifies the list form of the MODESET macro instruction.

MODESET (Execute Form)

The execute form of the MODESET macro instruction that generates an SVC is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede MODESET.
MODESET	
b	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 exception:

MF=(E,*list addr)*
specifies the execute form of the MODESET macro instruction, using a parameter list address.

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. Because the byte of storage exists in a multiprocessing environment, the possibility exists that the byte might be changed by another processor at the same time. Storage modification during NIL processing is accomplished by using the compare and swap (CS) instruction.

For details on the and immediate and compare and swap instructions, see *Principles of Operation.*

The NIL macro instruction is written as follows:

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

byte addr	*byte addr:* RX-type address.
,mask	*mask:* symbol or self defining term.
,REF=stor addr	*stor addr:* RX-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 0-15. *reg2:* symbol, or decimal digits 1-15. *reg3:* symbol, or decimal digits 0-15. **Default** for *reg1:* 0 **Default** for *reg2:* 1 **Default** for *reg3:* 2

The parameters are explained as follows:

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 4095. 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: Turn off bit ASCBXMET in byte ASCBCS1. The reference field, ASCBFW3, specifies the word being updated.

```
NIL ASCBCS1,X'FF'-ASCBXMET,REF=ASCBFW3
```

NUCLKUP - Nucleus Map Lookup Service

The NUCLKUP macro instruction can be used either to retrieve the address and AMODE of a nucleus CSECT or ENTRY or to retrieve the name and address of the nucleus CSECT, which is pointed to by a given address within the CSECT.

This macro runs in the key and state of the caller. On entry to this macro, register 13 must point to a 72-byte register save area.

The NUCLKUP macro instruction is written as follows:

name	*name:* symbol. Begin name in column 1.
b	One or more blanks must precede NUCLKUP.
NUCLKUP	
b	One or more blanks must follow NUCLKUP.
BYNAME,NAME=*name id* BYADDR,NAME=*name loc*	*name id:* 8-byte literal (enclosed in apostrophes), or the address of the 8-byte literal which can be either an RX-type address, or register (1) - (12). *name loc:* RX-type address or register (1) - (12).
,ADDR=*addr*	*addr:* RX-type address, or register (0) or (2) - (12).

The parameters are explained as follows.

BYNAME
BYADDR
specifies the function to be performed. If BYNAME is specified, the user supplies the name of a CSECT or ENTRY and receives the address and AMODE of that CSECT or ENTRY. If BYADDR is specified, the user supplies an address within a CSECT and receives the name and address of the CSECT.

,NAME=*name id*
,NAME=*name loc*
specifies the name or the location of the name of the CSECT depending on the option requested. If the user specifies BYNAME, *name id* contains the 8-character name to be searched for or the address of that name. If the user specifies BYADDR, *name loc* will contain the address of the 8-byte area in which the CSECT name is to be returned.

,ADDR = *addr*

contains the address to be searched for if BYADDR is specified; contains the address of the CSECT or ENTRY that is returned if BYNAME is specified.

The NUCLKUP service routine sets bit 0 of the word containing the address returned on a BYNAME request to indicate the AMODE. For example, if the requestor's AMODE is 31-bit and the AMODE of the CSECT is ANY, the NUCLKUP service routine sets bit 0 to 1. The setting of bit 0 is summarized in the following table:

Requestor's AMODE	AMODE of CSECT		
	24	31	ANY
24	0	1	0
31	0	1	1

When control is returned, the registers contain the following information:

Register	Meaning
0	For a BYNAME request, the address and AMODE of the CSECT or ENTRY; for a BYADDR request, the 31-bit address of the CSECT
1	For a BYNAME request, the high-order byte is zero and the low-order three bytes contain the length from the entry point to the end of the CSECT; for a BYADDR request, unchanged
2-14	Unchanged
15	Return code

The return codes in register 15 are as follows:

Hexadecimal Code	Meaning
0	The request was satisfied.
4	The request was not satisfied.
	For a BYNAME request, the name was not found and the location containing the address w set to zero.
	For a BYADDR request, the address was not found in the nucleus and the location containi the name was set to zero.
8	The request was not satisfied because the type of request was not specified correctly. The locations containing the name and address were set to zero.

Example 1

Operation: Place the address and AMODE of entry point IEAVESTU in register 0.

```
NUCLKUP BYNAME,NAME='IEAVESTU',ADDR=(0)
```

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. Because the byte of storage exists in a multiprocessing environment, the possibility exists that the byte might be changed by another processor at the same time. Storage modification during OIL processing is accomplished by using the compare and swap (CS) instruction.

For details on the or immediate and compare and swap instructions, see *Principles of Operation.*

The OIL macro instruction is written as follows:

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

byte addr	*byte addr:* RX-type address.
,mask	*mask:* symbol or self defining term.
,REF=*stor addr*	*stor addr:* RX-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 0-15. *reg2:* symbol, or decimal digits 0-15. *reg3:* symbol, or decimal digits 0-15. **Default** for *reg1:* 0 **Default** for *reg2:* 1 **Default** for *reg3:* 2

The parameters are explained as follows:

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 4095. 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: Turn on bit ASCBXMET in byte ASCBCS1. The reference field ASCB specifies the area containing the word being updated.

```
OIL ASCBCS1,ASCBXMET,REF=ASCB
```

PCLINK - Stack, Unstack, or Extract Program Call Linkage Information

Routines that receive control as a result of a PC instruction use the PCLINK macro instruction to provide a standardized method of maintaining PC linkage information. PCLINK has three forms:

- PCLINK STACK saves some of the environment when a routine gets control as a result of a PC instruction.
- PCLINK UNSTACK restores that environment before the routine issues a PT instruction to return control to the calling routine.
- PCLINK EXTRACT retrieves information from the saved environment.

STACK Option of PCLINK

To use PCLINK STACK you must be in primary mode and supervisor state. You must not change registers 13-4 between the time you get control and the time you issue PCLINK STACK.

The STACK option of the PCLINK macro instruction is written as follows:

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

STACK	
,INKEY=ZERO	
,OUTKEY=CALLER ,OUTKEY=ZERO ,OUTKEY=KEYn	**Default:** OUTKEY=CALLER KEYn: Any valid PSW key value where n = 0-F.
,SAVE=YES ,SAVE=NO	**Default:** SAVE=YES
,RELATED=*value*	*value:* any valid macro keyword specification.

STACK,INKEY = ZERO
specifies that the PSW key is zero upon entry to PCLINK. If this parameter is not specified, the macro expansion temporarily changes the PSW key to zero.

,OUTKEY = CALLER
,OUTKEY = ZERO
,OUTKEY = KEYn where n is 0-F
specifies the setting of the PSW key after the PCLINK macro instruction has completed. Specifying CALLER causes the PSW key to be restored to the value it had on entry. Specifying ZERO sets the PSW key to zero. Specifying a key value indicates a specific value for the key.

,SAVE = YES
,SAVE = NO
specifies whether (YES) or not (NO) to preserve registers 8 - 12. The save area used is different from the area addressed by register 13. SAVE = YES is the default. Processing is more efficient if you code SAVE = NO.

,RELATED = *value*
specifies information used to self-document macro instructions by "relating" functions or services to corresponding services performed elsewhere. The format and contents of the information specified can be any valid coding values.

On completion of PCLINK STACK, the registers are as follows:

R0, R1	Unchanged
R2	Bits 0-23 contain bits 8-31 from register 2 at the time the macro was issued. Bits 24-31 contain the PCLINK caller's PSW key.
R3, R4	Unchanged
R5	Linkage register to return from PCLINK STACK
R6, R7	Unchanged
R8-R12	Unchanged if SAVE = YES Unpredictable if SAVE = NO
R13	0, to ensure that the first save area created after the PC does not point to a previous save area.
R14	Stack token to uniquely identify the stack entry created. This token is required for the UNSTACK and EXTRACT forms of PCLINK.
R15	Unchanged

UNSTACK Option of PCLINK

To use PCLINK UNSTACK, you must be in supervisor state. In addition, if you specify PCLINK UNSTACK,THRU and the token contained in the specified register indicates the stack element most recently queued for that unit of work, you must be in primary mode and the PASID must be the same as when the stack element was created.

The UNSTACK option of the PCLINK macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede PCLINK.
PCLINK	
b	One or more blanks must follow PCLINK.
UNSTACK	
,THRU=*(reg)* ,TO=*(reg)* ,PURGE=YES	*reg:* Register (0) - (15).
,INKEY=ZERO	
,OUTKEY=STACK ,OUTKEY=ZERO	**Default:** OUTKEY=STACK
,SAVE=YES ,SAVE=NO	**Default:** SAVE=YES
,ERRET=*addr*	*addr:* RX-type address or register (0) - (13) or (15).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

UNSTACK,THRU= *(reg)*

specifies that the stack element identified by the token contained in the specified register, as well as all more recently stacked elements, are to be removed from the requestor's stack. The stack element specified by the token is used to restore registers. If the system cannot process the request, the routine specified by the ERRET parameter gets control; if the ERRET parameter is not specified, the requestor is abnormally terminated.

Processing is more efficient if you issue a separate PCLINK UNSTACK,THRU for each stack element you want to dequeue rather than unstacking several elements at a time.

If the token you specify represents the most recently enqueued stack element, the PASID when UNSTACK,THRU is issued must be the same as the PASID when PCLINK STACK was issued for that element.

When a PCLINK UNSTACK,THRU is completed, the PSW program mask is restored from the stack element identified by the token and the registers are as follows:

R0-R1	Unchanged
R2	Bits 24-27 contain the PSW key from the stack element identified by the token
R3	As saved by PCLINK STACK
R4-R7	Unchanged
R8-R12	Unchanged if SAVE=YES is specified Unpredictable if SAVE=NO is specified
R13,R14	As saved by PCLINK STACK
R15	Unchanged

,TO= *(reg)*

specifies that all stack elements stacked more recently than the element identified by the token contained in the specified register are to be removed from the stack. The element identified by the token remains on the stack. If the system cannot process the request, the routine specified by the ERRET parameter gets control; if the ERRET parameter is not specified, the requestor is abnormally terminated.

Use the TO parameter for stack cleanup in an FRR or ESTAE retry routine or in an FRR that is going to retry.

When a PCLINK UNSTACK,TO is completed, the registers are as follows:

R0,R1	Unpredictable
R2	Unchanged if INKEY=ZERO is specified and ERRET is not specified, otherwise, PSW key of PCLINK caller
R3-R7	Unchanged
R8-R12	Unchanged if SAVE=YES is specified Unpredictable if SAVE=NO is specified
R13	Unchanged
R14-R15	Unpredictable

,PURGE=YES

specifies that each stack element is to be freed until no more exist on the requestor's stack. Any element that resides in a terminated address space as well as elements stacked prior to it are not freed, but the stack pointer indicates an empty stack and the PCLINK request returns normally to the caller.

The ERRET parameter cannot be used with PURGE.

When the PCLINK UNSTACK,PURGE is completed, the registers are as follows:

R0,R1	Unpredictable
R2	Unchanged if INKEY=ZERO is specified, otherwise PSW key of PCLINK caller
R3-R7	Unchanged
R8-R12	Unchanged if SAVE=YES is specified Unpredictable if SAVE=NO is specified
R13	Unchanged
R14-R15	Unpredictable

,INKEY=ZERO

specifies that the PSW key is zero on entry to PCLINK. If this parameter is not specified, the macro expansion temporarily changes the key to zero.

,OUTKEY=STACK
,OUTKEY=ZERO

specifies the setting of the PSW key after the PCLINK request is completed. Specifying OUTKEY=ZERO returns to the caller in key zero. Specifying OUTKEY=STACK restores the key to the value contained in the stack element identified by token. OUTKEY=STACK is the default.

This parameter is valid only with PCLINK UNSTACK,THRU.

,SAVE=YES
,SAVE=NO

specifies whether (YES) or not (NO) registers 8 - 12 are to be preserved. The save area used for these registers is not the area pointed to by register 13.

,ERRET=*addr*

specifies the address of an exit routine to be given control if PCLINK UNSTACK encounters an error. ERRET is valid only with the TO and THRU parameters.

The ERRET exit routine receives control in the addressing mode of the caller of PCLINK. When an ERRET exit routine gets control, the cross memory state is the same as when the PCLINK macro instruction was issued. The registers are as follows:

R0,R1,R3,R13	Unpredictable
R2	PSW key of PCLINK caller
R4-R7	Unchanged
R8-R12	Unchanged if SAVE=YES is specified Unpredictable if SAVE=NO is specified
R14	The token passed as input
R15	4 - stack was empty 8 - input token is invalid 12 - an address on the STKE queue is invalid 16 - An ASID on the STKE queue is invalid 20 - Unknown error

,RELATED=*value*

specifies information used to self-document macro instructions by "relating" functions or services to corresponding services performed elsewhere. The format and contents of the information specified can be any valid coding values.

EXTRACT Option of PCLINK

To use PCLINK EXTRACT, you must either be in PSW key 0, supervisor state, or have a PSW key mask authorized for key 0.

In addition, you must have addressability to the same address space as when PCLINK STACK was issued for the stack element from which you are extracting data.

PCLINK EXTRACT modifies registers 0, 1, 14, and 15. If ALL=YES is specified, registers 13-4 are also modified.

The EXTRACT option of the PCLINK macro instruction is written as follows:

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

EXTRACT	
,TOKEN=*(reg)*	*reg:* Register (0) - (15).
,ALL=YES ,SVAREA=*(reg)* ,RETADR=*(reg)* ,PARM15=*(reg)* ,PARM0=*(reg)* ,PARM1=*(reg)* ,KEY=*(reg)* ,ASID=*(reg)* ,LP=*(reg)* ,ENTRY=*(reg)*	
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

EXTRACT,TOKEN= *(reg)*
specifies a register that contains a 32-bit stack token identifying the most recently stacked element.

,ALL=YES
specifies that all information stored in the stack element identified by the token is to be extracted. The stored information is placed into the same registers (registers 13, 15, and 0-4) it was in when PCLINK STACK was issued. Registers 5 and 14 are not restored.

,SVAREA= *(reg)*
specifies a register into which the address of the program call issuer's save area is to be placed.

,RETADR = *(reg)*
specifies a register into which the AMODE (in which control is to be returned), the return address, and PSW problem state bit are to be placed. These occupy bits 0,1-30, and 31, respectively.

,PARM15 = *(reg)*
,PARM1 = *(reg)*
,PARM0 = *(reg)*
specifies a register into which the contents of register 15 (PARM15), register 1 (PARM1), or register 0 (PARM0) at the time PCLINK STACK was issued are to be placed.

,KEY = *(reg)*
specifies a register into which the PC issuer's PSW key is to be placed. The key occupies bit positions 24-27, which are the same positions as those used by the IPK instruction.

,ASID = *(reg)*
specifies a register into which the PC issuer's PSW key mask (bits 0-15) and ASID (bits 16-42) are to be placed.

,LP = *(reg)*
specifies a register into which the latent parameter pointer is to be placed.

,ENTRY = *(reg)*
specifies a register into which the contents of register 5 as established by the PCLINK STACK macro instruction are to be placed. Bit 0 of the register used by the ENTRY parameter specifies the addressing mode of the program call routine that issued the PCLINK macro instruction.

,RELATED = *value*
specifies information used to self-document macro instructions by "relating" functions or services to corresponding services performed elsewhere. The format and contents of the information specified can be any valid coding values.

PGANY - Page Anywhere

Some fixed pages are assigned within the first 16 megabytes of storage. The real storage manager (RSM) assumes that once a page has been fixed, it is likely to be fixed again. Therefore, RSM sets a bit to indicate that a page was previously fixed and required storage in the first 16 megabytes of real storage. The next time that page is loaded, RSM tries to put it in the first 16 megabytes in anticipation of a fix. Use the PGANY macro instruction to indicate to RSM that no further page fixes are planned for a particular page and that the next time the page is loaded, RSM can put it anywhere.

Entry is by means of an SVC. The caller can be in either problem or supervisor state and must not hold any locks. On entry, register contents are as follows:

Register 0	-	Zero
Register 1	-	If bit 0 of byte 0 is 1, register 1 contains a pointer to the virtual subarea list. If bit 0 of byte 0 is 0, registers 1 and 15 contain a virtual subarea list entry.

On return, register contents are as follows:

Registers 0-1	Unpredictable
Registers 2-14	Unchanged
Registers 15	Return code

The PGANY macro instruction is written as follows:

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

L,LA=*list addr*	*list addr:* RX-type address or register (1) or (2) - (12).
R,A=*start addr*	*start addr:* RX-type address or register (1), (2) - (12).
,EA=*end addr*	*end addr:* RX-type address or register (15), (2) - (12). **Note:** Cannot be specified unless R is specified. **Default:** EA=start addr + 1.

The parameters are explained as follows:

L

specifies that the virtual subarea list (VSL) is being supplied with this request. (See the topic "Input to Page Services" in Volume 1 for a description of the virtual subarea list.)

,LA=*list addr*
specifies the address of the virtual subarea list.

R

specifies that the necessary parameters will be passed in registers. A virtual subarea list is not being supplied.

,A=*start addr*
specifies the address of the start of the virtual area.

,EA=*end addr*
specifies the end + 1 byte of the virtual area. If this parameter is not coded, the default is the start address + 1.

Note: *start addr* and *end addr* must be located in 24-bit addressable storage.

Upon completion, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Operation completed normally.
04	Parameter error, X'171' abend, operation terminated because of invalid address in VSL entry.
10	Parameter error, X'171' abend, operation terminated abnormally because the VSL list was invalid.
14	Environmental error, X'028' abend.

For return codes 04 and 10, registers are loaded before the abend as follows:

R0	Unpredictable
R1	Abend code
R2-R10	Unpredictable
R11	Address of input VSL list or 0 for R-form
R12	0 (ECB address =0)
R13-R14	Current VSL entry being processed
R15	Return code

PGFIX - Fix Virtual Storage Contents

The PGFIX macro instruction makes virtual storage areas, below 16 megabytes, resident in real storage and ineligible for page-out while the requesting task's address space occupies real storage. The PGSER macro instruction performs this function for addresses either above or below 16 megabytes. PGFIX (and PGSER) ignore requests to fix storage in a system area that has the fixed attribute (for example, the LSQA and SQA). A FIX request for a page in the LSQA or SQA will not cause the page to be backed by real storage below 16 megabytes. A subsequent PGFREE is effective only if issued by the same task. 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.
b	One or more blanks must precede PGFIX.
PGFIX	
b	One or more blanks must follow PGFIX.

R L	
,LA=*list addr*	*list addr:* A-type address, or register (1) or (2) - (12).
,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 as follows:

R

specifies that no parameter list is being supplied with this request.

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 (VSL). See the topic "Input to Page Services" in Volume 1 for a description of the VSL.

,A=*start addr*

specifies the start address of the virtual area to be fixed.

Note: *start addr* must be located in 24-bit addressable storage.

,ECB=*ecb addr*

specifies the address of the ECB that is used to signal event completion. If the ECB address specified is zero, (ECB=0 or ECB=(register) where the contents of the register specified is 0), the fix request is completely satisfied before control is returned.

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.

Note: *end addr* must be located in 24-bit addressable storage.

,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) before the fix is done.

,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.

Upon completion, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Operation completed normally; ECB posted complete.
04	Operation abnormally terminated with a X'171' abend. Operation incomplete because of invalid address virtual subarea list entry; ECB posted complete. See *Message Library: System Codes* for a complete description of the register contents after a X'171' abend.
08	Operation proceeding; ECB will be posted when all requested pages are fixed in real storage.
10	Operation abnormally terminated with a X'171' abend. Virtual subarea list entry or ECB address invalid; no ECB is posted. See *Message Library: System Codes* for a complete description of the register contents after a X'171' abend.

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.

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 is actually fixed. The storage is long fixed.

```
PGFIX R,A=(R3),ECB=(R5)
```

Example 2

Operation: Fix virtual storage without using a virtual subarea list. Storage is 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 forfeited before fixing the area specified by registers 3 and 4.

```
PGFIX R,A=(R3),EA=(R4),ECB=(R5),LONG=N,RELEASE=Y
```

PGFIXA - Fix Virtual Storage Contents

The PGFIXA macro instruction makes virtual storage areas, below 16 megabytes, resident in real storage and ineligible for page-out while the requesting task's address space occupies real storage. The PGSER macro instruction performs this function for addresses either above or below 16 megabytes. The PGFIXA function is available only to key zero and supervisor state users. The PGFIXA macro instruction executes short-term, synchronous page fixes. The preferred area(s) of storage are intended for long term page fixes. A long term page fix in the V=R or non-preferred areas may delay V=R functions or CONFIG STORAGE commands. All fix processing is assumed to be short-term and is complete when control is returned to the issuer of the macro.

PGFIXA does not prevent pages from being paged out when an entire address space is swapped out of real storage. Consequently, when using the PGFIXA macro instruction, you cannot assume a constant real address mapping for fixed pages that are susceptible to swapping.

Output

If the PGFIXA is successful, control is returned enabled to the user, all pages are fixed, and register 15 contains a return code of zero.

If the PGFIXA is unsuccessful, the user will be abended with a system completion code of X'171' or a system complete code of 028. For X'171' abends, all pages processed up to, but not including the page causing the error, will be fixed. Register 10 will contain the address of the pages in error when the abend is issued. No pages will be fixed in the event of a X'028' abend.

Restrictions

Use of the PGFIXA macro instruction is subject to the following restrictions:

- Can be used only for short term synchronous fixes.
- The user must be in supervisor state with a protection key of zero.
- The user must not hold any spin locks.
- The program mask byte in the PSW is zero and interrupts are enabled upon return from the PGFIXA.
- The user is responsible for freeing any pages fixed via the PGFIXA. A corresponding PGFREEA macro instruction should be issued. In addition, an FRR should be established during the period where fixes are outstanding. The FRR should free the frames in case there is an unexpected error.
- DSECTs for the IHAPSA, CVT, and IHAPVT must be provided.

- The user must ensure that the end address is greater than or equal to the start address.
- The SAVE keyword can only be used with TYPE=R.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede PGFIXA.
PGFIXA	
b	One or more blanks must follow PGFIXA.
,TYPE=L ,TYPE=R	**Default:** TYPE=R
,SAVE=YES ,SAVE=NO	**Default:** SAVE=YES

The parameters are explained as follows:

TYPE=L
TYPE=R
specifies the type of input. When L is specified, register 1 is to contain the address of a virtual subarea list (VSL) fixed in storage. (See the topic "Input to Page Services" in Volume 1 for a description of the VSL.) By specifying TYPE=L, registers 1 through 13 are saved. If TYPE=R is specified, then register 1 contains the address of the first byte to be fixed in a contiguous range and register 2 contains the address of the last byte to be fixed (actual end address). When TYPE=R is specified, the registers saved depend upon what is specified on the SAVE parameter.

Note: All other users of the PGFIX, PGFIXA (TYPE=L), and PGFREEA macro instructions must specify the actual end address plus one.

,SAVE=YES
,SAVE=NO
specifies the registers to be saved for TYPE=R. Registers 1 through 13 are saved if SAVE=YES is specified or if the default is taken. Registers 2 through 10 are saved if SAVE=NO is specified.

Example 1

Operation: Use PGFIXA to fix virtual storage without using a virtual subarea list. Registers 2 through 10 will be saved.

```
FIX1 PGFIXA TYPE=R,SAVE=NO
```

Example 2

Operation: Use PGFIXA to fix virtual storage using a virtual subarea list. Registers 1 through 13 will be saved.

```
FIX2 PGFIXA TYPE=L
```

PGFREE - Free Virtual Storage Contents

The PGFREE macro instruction makes virtual storage pages, below 16 megabytes, that were fixed via the PGFIX macro instruction eligible for page-out. The PGSER macro instruction performs this function for addresses either above or below 16 megabytes. The PGFREE function is available only to authorized system functions and users. PGFREE must be issued by the same task that issued the PGFIX, otherwise PGFREE has no effect.

Note: 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 made pageable as the result of issuing a PGFREE macro instruction.

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

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

L	
,LA=*list addr*	*list addr:* A-type address, or register (1) or (2) - (12).
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
,ANYWHER=N ,ANYWHER=Y	**Default:** ANYWHER=N
,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 as follows:

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 (VSL). See the topic "Input to Page Services" in Volume 1 for a description of the VSL.

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.

Note: *start addr* must be located in 24-bit addressable storage.

,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.

Note: *end addr* must be located in 24-bit addressable storage.

,ANYWHER=N
,ANYWHER=Y
On subsequent page-ins, assign real storage frames below 16 megabytes in anticipation of a page fix (N) or on subsequent page-ins, assign real storage frames anywhere (Y). The ANYWHER option takes effect only when the page fix count goes to zero. The default is ANYWHER=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.

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.

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 fully included in the address range.

```
PGFREE R,A=(R3),EA=(R4),ECB=(R5),RELEASE=Y
```

PGFREEA - Free Virtual Storage Contents

The PGFREEA macro instruction makes virtual storage areas, below 16 megabytes, that were fixed by the PGFIXA macro instruction eligible for page-out. The PGSER macro instruction performs this function for pages either above or below 16 megabytes. The PGFREEA function is available only to key zero and supervisor state users.

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

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

No additional parameters are specified.

Restrictions

Use of the PGFREEA macro instruction is subject to the following restrictions:

- The issuer of the PGFREEA must provide a fixed virtual subarea list (VSL) or chain of them, pointed to by register 1.
- The user must be in supervisor state, protection key 0.
- The user must provide DSECTs for IHAPSA, CVT, and IHAPVT.

Output

If the PGFREEA is successful, all pages will be freed and register 15 will contain a return code of zero. If unsuccessful, all pages up to, but not including the one that caused the abend will be freed. The user will be abended with a system completion code of X'171'.

PGSER - Page Services

The PGSER macro instruction and its fast path version perform the same paging services that PGANY, PGFIX, PGFIXA, PGFREE, PGFREEA, PGLOAD, PGOUT, and PGRLSE perform for addresses below 16 megabytes. PGSER performs these services for addresses either above or below 16 megabytes.

Except for the TCB, all input parameters to this macro instruction can reside in storage above 16 megabytes if the caller is executing in 31-bit addressing mode.

The services are:

- Page fix equivalent to PGFIX
- Fast path to fix virtual storage
- Page free equivalent to PGFREE
- Fast path to free virtual storage
- Page load equivalent to PGLOAD
- Page out equivalent to PGOUT
- Page release equivalent to PGRLSE
- Page anywhere equivalent to PGANY

This macro is also described in *Supervisor Services and Macro Instructions* with the exception of the restricted parameters. The parameters FIX and FREE are restricted to APF-authorized, key zero, or supervisor state callers. The parameters BRANCH=SPECIAL and BRANCH=Y are restricted to enabled, supervisor state, key zero callers; users of these options must provide the address of an 18-word save area in register 13. (See the section "Branch Entry to the PGSER Routine" in volume 1 for more information about branch entry.) The RELEASE option of the macro is restricted to supervisor state key zero users if the common area is being released. Non-authorized users can release only the private area.

Regardless of the addressing mode, all addresses passed in registers are used as 31-bit addresses. All RX-type addresses are assumed to be in the addressing mode of the caller.

The syntax of the fast path version of PGSER is presented separately following this standard description. The standard form of the PGSER macro instruction is written as follows:

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

R L	
,FIX ,FREE ,LOAD ,OUT ,RELEASE ,ANYWHER	
,LA=*list addr*	*list addr:* RX-type address or register (1), (5) - (12) for branch entry; or register (1), (2) - (12) for SVC entry. **Note:** This parameter is valid only with L.
,A=*start addr*	*start addr:* RX-type address or register (1), (5) - (12) for branch entry; or register (1), (2) - (12) for SVC entry. **Note:** This parameter is valid only with R.
,EA=*end addr*	**Default:** EA=*start addr* *end addr:* RX-type address or register (2), (5) - (12) for branch entry; or register (15), (2) - (12) for SVC entry. **Note:** This parameter is valid only with R.
,TCB=*tcb addr*	**Default:** TCB=0 *tcb addr:* RX-type address or register (4), (5) - (12). **Note:** This parameter can be specified only if FIX, FREE, LOAD, or OUT and BRANCH=Y are specified.
,ECB=*ecb addr*	**Default:** If FREE or LOAD is specified, ECB=0. **ecb addr:** RX-type address or register (0), (5) - (12) for branch entry; or register (0), (2) - (12) for SVC entry. **Note:** This parameter is required if FIX is specified; is optional if FREE or LOAD is specified; and is invalid for OUT, RELEASE, or ANYWHER. For synchronous page fix the ECB address must be 0.
,RELEASE=Y ,RELEASE=N	**Default:** RELEASE=N **Note:** This parameter may be specified only if FIX, FREE, or LOAD is specified.
,LONG=Y ,LONG=N	**Default:** LONG=Y **Note:** This parameter may be specified only if FIX is specified.
,BACKOUT=Y ,BACKOUT=N	**Default:** BACKOUT=Y **Note:** This parameter may be specified only if FIX is specified.
,KEEPREL=Y ,KEEPREL=N	**Default:** KEEPREL=N **Note:** This parameter may be specified only if OUT is specified.
,ANYWHER=Y ,ANYWHER=N	**Default:** ANYWHER=N **Note:** This parameter may be specified only if FREE is specified.
,BRANCH=Y ,BRANCH=N	**Default:** BRANCH=N
,RELATED=*value*	*value:* any valid macro keyword specification.

R
L

specifies the manner in which the input is supplied. If R is specified, the user supplies the starting and ending addresses of the virtual area for which the service needs to be performed. Before processing the request, page services puts these addresses in registers 1 and 15, respectively. If L is specified, the user supplies the address of the page services list (PSL), which specifies the virtual area for which the service is to be performed. Before processing the request, page services puts the address of the PSL in register 1. See the topic "Input to Page Services" in Volume 1 for a description of the PSL.

,FIX
,FREE
,LOAD
,OUT
,RELEASE
,ANYWHER

indicates the function to be performed.

FIX specifies that the virtual storage areas are to reside in real storage and are ineligible for page-out while the address space is swapped in. This parameter does not prevent pages from being paged out when the entire address space is swapped out of real storage. FIX will ignore a request to fix storage in a system area that has the fixed attribute (for example, the LSQA and SQA). A FIX request for a page in the LSQA or SQA will not cause the page to be backed by real storage below 16 megabytes.

FREE specifies that the virtual storage areas that were previously fixed via the FIX option are eligible for page-out. A fixed page is not considered pageable until the number of FREE and FIX requests for the page are equal.

LOAD specifies that a page-in operation is to be initiated for the virtual storage area specified, in anticipation of future needs.

OUT specifies that a page-out operation is to be initiated for the virtual storage area specified.

RELEASE specifies that all real and auxiliary storage, associated with the virtual storage area specified, is to be released.

ANYWHER applies to virtual storage areas that did not specify LOC=(BELOW,ANY) or LOC=(ANY,ANY) or LOC=ANY on a GETMAIN request, that have been previously fixed, and probably will not need to be fixed again. ANYWHER specifies that the virtual storage area specified can be placed either above or below 16 megabytes real on future page-ins.

,LA=*list addr*

specifies the address of the page services list (PSL) for L requests.

,A=*start addr*

specifies the address of the start of the virtual area for R requests.

,EA=*end addr*

specifies the address of the end of the virtual area for R requests.

,TCB=*tcb addr*

specifies either zero or the address of the TCB to be assigned ownership of fixes for a FIX request or fixes for a FREE request. If zero is specified, no TCB is assigned ownership of the request. Cross memory callers must specify zero.

For OUT and LOAD requests, the PGSER routine associates the request with a particular TCB so that the request can be purged if the task terminates before the request is complete. For SVC entry (BRANCH=N), the PGSER routine uses the current TCB.

Note: The TCB resides in storage below 16 megabytes.

,ECB=*ecb addr*

specifies the address of the ECB that is used to signal event completion for an asynchronous FIX or LOAD request. If the caller is in cross memory mode or if the caller requests a synchronous page fix (a FIX for which the caller is suspended until the entire FIX request is complete), the ECB must be zero (ECB=0 or ECB=(r), where (r) represents a register that contains zero).

For a FREE request, ECB specifies the address of the ECB that was used in a previous FIX request. If this parameter is specified, any pages in the previous FIX request that are not yet fixed, will not be fixed. If L is specified, the PSL chain must contain the addresses of the virtual pages in the same order in both the FREE and the previous FIX request. Also, the ECB for the FIX request will not be posted if it was not yet posted at the time of the FREE request.

If the ECB parameter is not specified on a FREE request, only the fix counts for the valid pages in storage at the time of the FREE request are decreased. This will not affect the paging activity and the posting of the ECB associated with the original FIX request.

If an ECB is supplied on a FIX or LOAD request, the caller must check the return code because the ECB will not be posted if the return code is zero. If an ECB is not supplied, it is not necessary to check the return code because control returns to the caller only if the request was successfully completed; if unsuccessful, page services abnormally terminates the caller.

For all callers that supply an ECB, page services verifies that the ECB address is in an area allocated via GETMAIN and if the caller is not in key 0, page services also verifies that the ECB is in the caller's protect key. Before posting the ECB, page services again verifies that the ECB is located in an allocated area and that the ECB is in the caller's protect key. (This is to check that the allocated area has not been freed via FREEMAIN and the protect key has not been changed.) It is the user's responsibility to ensure that the page containing the ECB is not freed and that the key is not altered. If either test fails, page services does not post the ECB.

,RELEASE=Y
,RELEASE=N

specifies that all the real and auxiliary storage associated with the virtual storage areas is to be released to the system (Y) or that all the real and auxiliary storage associated with the virtual storage areas is not to be released to the system (N).

,KEEPREL=Y
,KEEPREL=N
specifies that the virtual pages should be validated again after the page-out completes (Y); or that the virtual pages will be marked invalid and the real storage frames freed for reuse (N).

,LONG=Y
,LONG=N
specifies that the relative real time anticipated for the FIX is long (Y); or that the relative real time anticipated for the FIX is short (N). (In general, the duration of a fix is long if it can be measured in seconds.)

,BACKOUT=Y
,BACKOUT=N
specifies the procedure to follow when a non-allocated page is encountered during the processing of a FIX request. If BACKOUT=Y, all pages fixed as part of the request are freed before returning to the caller. If BACKOUT=N, the pages previously fixed as part of the request are not freed and no further processing is done before returning to the caller.

,ANYWHER=N
,ANYWHER=Y
specifies that on subsequent page-ins, page services is to assign real storage frames below 16 megabytes in anticipation of a page-fix (N); or on subsequent page-ins, page services is to assign real storage frames anywhere (Y). The ANYWHER option takes effect only when the page-fix count goes to zero.

,BRANCH=Y
,BRANCH=N
specifies whether or not this is a branch entry.

If BRANCH=Y is specified, it is a branch entry; and users of this option must provide the address of an 18-word save area in register 13. Register 2 contains the ending address.

If BRANCH=N is specified, it is an SVC entry. Register 15 contains the ending address.

Cross memory callers must use BRANCH=Y.

,RELATED=*value*
provides information to document the macro by relating the service performed to some corresponding function or service. The format can be any valid coding value that the user chooses.

On return the register contents are as follows:

Register	Contents
0-4	The contents are destroyed and unpredictable.
5-13	The contents are unchanged.
14	The contents are destroyed and unpredictable.
15	This register contains the return code.

The return codes, given in register 15, along with the option used and the meaning follow:

Option	Code	Meaning
FIX	0	The operation completed normally and the ECB will not be posted.
FIX	8	The operation is proceeding, the ECB (if available) will be posted with X'00' when the requested pages are fixed.
FREE	0	The operation completed normally.
LOAD	0	The operation completed normally and the ECB will not be posted. If no ECB is supplied, the operation is completed or proceeding.
LOAD	8	The operation is proceeding. The ECB will be posted with X'00' when all page-ins are complete.
OUT	0 C	The operation completed normally. At least one page in the requested range was not paged out.
RELEASE	0	The operation completed normally.
ANYWHER	0	The operation completed normally.

If a error is found in one of the parameters, the requestor is abnormally terminated with a system abend code of X'18A' and one of the following hexadecimal reason codes is provided in register 15:

Hexadecimal Code	Meaning
4	A page-fix operation abnormally terminated cause of an invalid address in a PSL entry. The ECB will not be posted.
4	A page-release operation abnormally terminated because either a page release was attempted for permanently backed storage or a non-system key caller attempted to release storage in a different key.
10	A page-fix, page-free, or a page-load operation abnormally terminated because the PSL or ECB address was invalid.

Callers not authorized to use a specific service are abnormally terminated with a system abend code of X'28A' and a hexadecimal error code of X'10' in register 15. If an environmental error is encountered while processing the page-services request, the caller is abnormally terminated with a system abend code of X'028' and a hexadecimal error code of X'14' in register 15. A unique reason code is also provided in register 0 for these errors.

Example 1

Operation: Synchronously fix the page that starts at the address given in register 1 and ends at the address given in LOADWORD. Use branch entry. No particular TCB is associated with this request.

```
PGSER R,FIX,A=(1),ECB=0,EA=LOADWORD,TCB=0,BRANCH=Y
```

Example 2

Operation: Free the page specified in the PSL pointed to by register 2. The ECB address is given in register 8. Use branch entry. Release all real and auxiliary storage associated with this virtual area. Do not attempt to back the area below 16 megabytes on future page-ins.

```
PGSER L,FREE,LA=(2),ECB=(8),RELEASE=Y,ANYWHER=Y,BRANCH=Y
```

Example 3

Operation: Load the page specified in the PSL pointed to by register 1. Supply an ECB of zero.

```
PGSER L,LOAD,LA=(1),ECB=0
```

Example 4

Operation: Perform a page-out for the virtual area starting at the address given in register 1 and ending at the address given in register 5. The address of the TCB is given in register 8. Use branch entry.

```
PGSER R,OUT,A=(1),EA=(5),TCB=(8),BRANCH=Y
```

Example 5

Operation: Perform a page-out for the virtual area specified in the PSL located at LOADWORD. Use branch entry.

```
PGSER L,OUT,LA=LOADWORD
```

PGSER - Fast Path Page Services

The fast path PGSER macro instruction performs FIX and FREE requests for users on performance paths. The following restrictions apply to this special fast path service:

- Short term fixes only
- No ECB
- No TCB
- No VIO window pages
- Key 0, supervisor state callers only
- Enabled
- Register 13 must point to an 18-word save area in non-pageable storage.
- If the list format of the macro instruction is used, all user-defined short page service lists (SSLs) must be valid in nonpageable storage.

The fast path PGSER macro does not verify any of the restricted conditions. It is the user's responsibility to verify the restricted conditions and to provide recovery to purge FIX requests when the task terminates before a page service request is complete.

The fast path PGSER macro is written as follows:

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

R L	
,FIX ,FREE	
,LA=*list addr*	*list addr:* RX-type address or register (1), (5) - (12). **Note:** This parameter is valid only if L is specified.
,A=*start addr*	*start addr:* RX-type address or register (1), (5) - (12). **Note:** This parameter is valid only if R is specified.
,EA=*ending addr*	*ending addr:* RX-type address or register (2), (5) - (12). **Note:** This parameter is valid only if R is specified.
,BACKOUT=Y ,BACKOUT=N	**Default:** BACKOUT=Y **Note:** This parameter is valid only for FIX requests.
,ASCB=*addr*	*addr:* RX-type address or register (5) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.
,BRANCH=SPECIAL	

The parameters are explained as follows:

R
L

specify the manner in which the input is supplied. If R is specified, the user supplies the starting and ending addresses of the virtual storage area for which the service is to be performed. Before processing the request, page services puts these addresses in registers 1 and 2, respectively. If L is specified, the user supplies the address of the short page services list (SSL), which specifies the virtual storage area for which the service is to be performed. Before processing the request, page services puts the address of the SSL in register 1. See the topic "Input to Page Services" in Volume 1 for a description of the SSL.

,FIX
,FREE

indicate the function to be performed.

FIX specifies that the virtual storage areas are to reside in real storage and are ineligible for page-out while the address space is swapped in. This parameter does not prevent pages from being paged out when the entire address space is swapped out of real storage. FIX will ignore a request to fix storage in a system area that has the fixed attribute (for example, the LSQA and SQA). A FIX request for a page in the LSQA or SQA will not cause the page to be backed by real storage below 16 megabytes

FREE specifies that the virtual storage areas that were previously fixed via the FIX option are eligible for page-out. A fixed page is not considered pageable until the number of FREE and FIX requests for the page are equal.

,LA = *list addr*
specifies the address of the short page service list (SSL) for L requests.

,A = *start addr*
specifies the address of the start of the virtual area for R requests.

,EA = *end addr*
specifies the address of the end of the virtual area for R requests.

,BACKOUT = Y
,BACKOUT = N
specify the procedure to follow if an unallocated page is encountered during the processing of a fix request.

If BACKOUT = Y is specified, all pages fixed as part of the request will be freed before returning to the caller.

If BACKOUT = N is specified, the pages previously fixed as part of the request will not be freed before returning to the caller. In this situation, no further pages are processed once an unallocated page is encountered.

,ASCB = *address of ASCB*
specifies the address of the ASCB for the currently addressable address space.

Note: The ASCB must reside in 24-bit addressable storage.

,RELATED = *value*
specifies information used to document the macro instruction and to relate the service performed to some corresponding service or function. The format of the information specified can be any valid coding values that the user chooses.

,BRANCH = SPECIAL
specifies a branch entry call to the fast path FIX and FREE services. If BRANCH = SPECIAL is specified, users must provide an 18-word save area in non-pageable storage.

Example 1

Operation: Fix 4096 bytes of storage starting at the address BUFFER. The address of the ASCB is in register 6.

```
PGSER R,FIX,A=BUFFER,EA=BUFFER+4095,BRANCH=SPECIAL,ASCB=(6)
```

Example 2

Operation: Free the area specified in the SSL defined at LISTSSL. Use the ASCB in PSAAOLD.

```
L    5,PSAAOLD
PGSER L,FREE,LA=LISTSSL,ASCB=(5),BRANCH=SPECIAL
```

POST - Signal Event Completion

Use the POST macro instruction to have the specified ECB (event control block) set to indicate the occurrence of an event. If this event satisfies the requirements of an outstanding WAIT or EVENTS macro instruction, the waiting task is taken out of the wait state and dispatched according to its priority. The bits in the ECB are set as follows:

```
Bit 0 of the specified ECB is set to 0 (wait bit).
Bit 1 is set to 1 (complete bit).
Bits 2 through 31 are set to the specified completion code.
```

The description of the POST macro instruction follows. The POST macro instruction is also described in *Supervisor Services and Macro Instructions*, with the exception of the ASCB, ERRET, ECBKEY, BRANCH, and MEMREL parameters. These parameters are restricted ir use to programs that are authorized (supervisor state, APF-authorized, or PSW key 0-7) and, therefore, are only described here. The BRANCH=YES parameter is further restricted to supervisor state and key 0.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede POST.
POST	
b	One or more blanks must follow POST.
ecb addr	*ecb addr:* RX-type address, or register (2) - (12), except (10).
,comp code	*comp code:* symbol, decimal or hexadecimal digit, or register (0), (2) - (9), (10), or (12). **Range of values:** 0 - 2^{30}-1 **Default:** 0
,ASCB=*addr*,ERRET=*err addr* ,ASCB=*addr*,,ERRET=*err addr*, ECBKEY=*key*	*addr:* RX-type address, or register (2) - (9), (12). *err addr:* RX-type address, or register (2) - (9), (12). *addr:* RX-type address, or register (2) - (9), (12). *err addr:* RX-type address, or register (2) - (9), (12). *key:* symbol, decimal or hexadecimal digit, or register (2) - (9), (12). **Range of values:** 0 - 15 (decimal) **Default:** None. **Note:** If the register form is specified, bits 24-27 of the register must contain the key.
,BRANCH=YES ,BRANCH=NO	**Default:** BRANCH=NO
,MEMREL=YES ,MEMREL=NO	**Default:** MEMREL=YES **Note:** MEMREL can be coded only if BRANCH=YES, and the ASCB and ERRET parameters are coded.
,RELATED=*value*	*value:* any valid macro keyword specification.

The explanation of the parameters is as follows:

ecb addr
specifies the address of the fullword event control block representing the event.

,comp code
specifies the completion code to be placed in the event control block upon completion.

,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 a POST failure is detected. See the topic "Cross Memory Post" in Volume 1 for information on the addressing mode in which the exit routine receives control.

Note: The ASCB must reside in 24-bit addressable storage

,ASCB=*addr* **,ERRET=***err addr* **,ECBKEY=***key*
specifies the address of the ASCB containing the ECB being posted, the address of the routine to be given control when an error condition resulting from a POST failure is detected, and the storage protection key of the ECB to be posted. If the ECB does not identify a current wait condition against it, the ECB is checked against the key before it is updated with the post completion code. Otherwise, the ECB is checked against the

protection key of the waiting task. (See the topic "Cross Memory Post" in Volume 1 for information about the addressing mode in which the exit routine receives control.)

Note: The ASCB must reside in 24-bit addressable storage

,BRANCH=YES
,BRANCH=NO
specifies branch entry (YES) or SVC entry (NO). The default is BRANCH=NO.

If BRANCH=YES is specified and the ASCB address is not specified, the caller must hold the local lock and be in non-cross memory mode. For branch entry callers, registers 0-9, 12, and 13 are preserved. For SVC callers, registers 2-14 are preserved.

If the caller specifies the ASCB address, holds the local lock of the home address space, and specifies MEMREL=YES (or allows it to default), then the current address space must be the home address space and registers 1-9 are preserved. If the ECBKEY parameter is not specified, register 0 is also preserved.

If the caller specifies the ASCB address and either does not hold home's local lock or has specified MEMREL=NO, only register 9 is preserved.

,MEMREL=YES
,MEMREL=NO
specifies whether the error routine specified by the ERRET parameter runs in the caller's address space (YES) or in the master scheduler's address space (NO). The default is MEMREL=YES.

If the caller holds the LOCAL lock of the home address space, MEMREL also indicates which registers the POST routine saves.

- If the LOCAL lock is held and MEMREL=YES is specified or defaulted to, registers 0 through 9 and register 14 are saved.

- If the LOCAL lock is not held or MEMREL=NO is specified, only register 9 and register 14 are saved.

If the LOCAL lock is held and MEMREL=YES, the current address space must be the home address space. The macro's results are unpredictable if the current address space is not the home address space.

If the LOCAL lock is not held or MEMREL=NO, the current address space can be any address space. Thus, when the cross memory mode and the lock status of the caller is unknown, specify MEMREL=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.

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.
b	One or more blanks must precede POST.
POST	
b	One or more blanks must follow POST.
ecb addr	*ecb addr:* A-type address.
,ASCB=*addr*,ERRET=*err addr* ,ASCB=*addr*,ERRET=*err addr*, ECBKEY=YES	*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.

,ASCB=*addr* **,ERRET=***err addr* **,ECBKEY=YES**
specifies that the storage protection key of the ECB is defined in the execute form of the POST macro instruction.

Note: The ASCB resides in 24-bit addressable storage.

POST (Execute Form)

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede POST.
POST	
b	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* ,ASCB=*addr*,ERRET=*err addr*, ECBKEY=*key*	*addr:* RX-type address, or register (2) - (12). *err addr:* RX-type address, or register (2) - (12). *key:* symbol, decimal or hexadecimal digit, or register (2) - (12). **Range of values:** 0 - 15 (decimal). **Default:** None. **Note:** If the register form is specified, bits 24-27 of the register must contain the key.
,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 exception:

,MF=(E,*prob addr)*
specifies the execute form of the POST macro instruction using a remote control program parameter list.

PROTPSA - Disable, Enable Low Address Protection

The PROTPSA macro instruction manipulates control register 0 to disable or enable low address protection. Low address protection requires programs that modify storage in address range 0-511 to disable low address protection before making the modification and then to enable low address protection after making the modification.

Restrictions

The PROTPSA macro instruction has the following restrictions:

- Users must include a DSECT for the PSA (via the IHAPSA mapping macro).
- PROTPSA must execute in supervisor state, protection key 0.
- I/O and external interrupts must be disabled while low address protection is disabled.
- The user must not call or transfer control to another program while low address protection is disabled.
- Low address protection should be disabled for a minimum amount of time.

The PROTPSA macro instruction is coded as follows:

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

DISABLE ENABLE	**Default:** None. **Note:** One or the other keyword must be specified.

The parameters are explained as follows:

DISABLE
specifies that low address protection will be disabled until a PROTPSA ENABLE instruction is executed.

ENABLE
specifies that low address protection will be enabled until a PROTPSA DISABLE is issued.

Example 1

Operation: Disable low address protection so that the restart new PSW can be modified and then enable low address protection.

```
PROTPSA DISABLE
MVC     FLCRNPSW,NEWPSW
PROTPSA ENABLE
```

PTRACE - Processor Trace

The PTRACE macro instruction creates a trace table entry and places it in the system trace table. The entry consists of an event identifier, the contents of a designated range of general registers or storage locations, and system supplied status information.

When using this macro, the user must provide the following information:

- The type of trace entry that is to be created
- The data to be recorded in the trace entry

The PTRACE macro instruction can only be issued with DAT-ON. The caller must be in key 0 and supervisor state but can be in cross memory mode and in either 24 or 31-bit addressing mode. All addresses passed to the PTRACE routine are treated as 31-bit addresses. PTRACE users must include the IHAPSA and IHATRVT mapping macros and register 13 must point to a 72-byte save area that can be used by the PTRACE service.

The PTRACE macro instruction is written as follows:

name	*name:* symbol. Begin in *name* in column 1.
b	One or more blanks must precede PTRACE.
PTRACE	
b	One or more blanks must follow PTRACE.
TYPE=USR*n*	*n:* hexadecimal digit 0 - F.
,REGS=(reg1,reg2) ,REGS=(1)	**Default:** REGS=(1) reg1: decimal digit 2 - 12. reg2: decimal digit 2 - 12.
,SAVEAREA=STANDARD	

The parameters are explained as follows:

TYPE=USR*n*
specifies a user-event explicit trace entry identified by the hexadecimal number *n*. Trace processing places this number in the trace entry for identification purposes.

,REGS = (*reg1,reg2*)
,REGS = (1)

defines the data to be placed in the user's trace entries. Multiple trace entries are created if more than 5 registers or 5 words of data are requested.

If REGS = (*reg1,reg2*) is specified, the data is located in a range of registers, where *reg1* specifies the first register in the range and *reg2* specifies the last register in the range. The register number, *reg2*, must always be greater than or equal to the register number, *reg1*. A maximum of 11 words of data can be indicated for tracing using REGS = (*reg1,reg2*).

If REGS = (1) is specified or used as the default, register 1 must contain the 31-bit address of a parameter list. The high order bit of this address must be set to 0. If REGS = (1) is specified, up to 1024 words of data can be recorded. The parameter list contains N + 1 fullword entries. The first word contains the number of words of data (N) to be recorded. This is followed by the N words of data to be placed in the user's trace entries.

,SAVEAREA = STANDARD

specifies that register 13 contains the 31-bit address of a 72-byte save area that can be used by the PTRACE routine.

When control is returned, registers 2-13 are restored to their original values, but the original contents of registers 0, 1, 14, and 15 are destroyed. On exit, register 15 contains the following return code:

Hexadecimal Code	Meaning
0	The function completed successfully.

Example 1

Operation: Create a trace table entry for user event 4. Registers 5, 6, and 7 contain the user data to be recorded.

```
PTRACE TYPE=USR4,REGS=(5,7),SAVEAREA=STANDARD
```

Example 2

Operation: Create trace table entries for user event C. Register 1 contains the address of a parameter list containing the data to be recorded.

```
PTRACE TYPE=USRC,REGS=(1),SAVEAREA=STANDARD
```

PURGEDQ - Purge SRB Activity

The PURGEDQ macro instruction allows 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 are unpredictable. For this reason, the PURGEDQ macro instruction is available to:

- Dequeue SRBs not yet dispatched.
- Allow processing for previously scheduled SRBs to complete.
- "Clean up" each dequeued SRB.

All parameters of this macro are optional. The parameters determine the target address space and limit the scope of the purge. When purging SRBs scheduled in the current space, PURGEDQ waits for dispatched SRBs to terminate. PURGEDQ does not purge or wait for terminations of SRBs scheduled into address spaces other than the current address space once they have been dispatched. The issuer of PURGEDQ is not informed of SRBs that cannot be purged.

Except for the TCB, all input parameters to this macro instruction can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede PURGEDQ.
PURGEDQ	
b	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 as follows:

RMTR = *RMTR addr*
If specified, limits the purge to SRBs that contain the same address in field SRBRMTR. This parameter has no effect on the execution of the resource manager termination routine specified in SRBRMTR.

,ASID = *ASID addr*
specifies the ASID of the address space (target address space) into which the SRB was scheduled. If omitted, the current address space is assumed.

,ASIDTCB = *TCB addr*
Provides a method of limiting the scope of the purge. ASIDTCB specifies the address of a double word that optionally defines a TCB address and/or ASID to match against fields SRBPTCB and SRBPASID respectfully. When non-zero values are specified, only SRBs that match the specified value will be purged. When the TCB address is non-zero, the ASID field must also be non-zero. If the ASIDTCB parameter is omitted, SRBPASID and SRBPTCB will be matched against the current address space ID and current TCB address. The double word has a format similar to field SRBFLC, which contains SRBPASID and SRBPTCB. The format of the double word is:

bytes 0-1	Reserved
bytes 2-3	ASID for match with SRBPASID or zero.
bytes 4-7	TCB address for match with SRBPTCB or zero. ASID must be specified is TCB address is specified.

Note: The TCB resides in storage below 16 megabytes.

Example 1

Operation: Purge all SRBs scheduled into the current address space, related to the current (terminating) task, and associated with the resource manager termination routine located at RESCLEAN.

```
PURGEDQ RMTR=RESCLEAN
```

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.
b	One or more blanks must precede PURGEDQ.
PURGEDQ	
b	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 exception:

,MF=L
specifies the list form of the PURGEDQ macro instruction.

Example 1

Operation: Specify the resource manager termination routine located at RESCLEAN and produce the parameter list to be used by the execute form of the PURGEDQ macro instruction.

```
STATPDQ PURGEDQ RMTR=RESCLEAN,MF=L
```

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.
b	One or more blanks must precede PURGEDQ.
PURGEDQ	
b	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 exception:

,MF=(E,*ctrl addr)*
specifies the execute form of the PURGEDQ macro instruction, using a remote control program parameter list.

Example 1

Operation: Purge all SRBs scheduled into the address space given in register 6 and associated with the resource manager termination routine located at RESCLEAN. Indicate that the remote control program parameter list is located at STATPDQ.

```
PURGEDQ ASID=(6),RMTR=RESCLEAN,MF=(E,STATPDQ)
```

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.
b	One or more blanks must precede QEDIT.
QEDIT	
b	One or more blanks must follow QEDIT.
ORIGIN = *CIB addr ptr*	*CIB addr ptr:* RX-type address, or register (0),(2) - (12).
,BLOCK = *CIB addr* ,CIBCTR = *CIB nmbr*	*CIB addr:* RX-type address, or register (1), (2) - (12). *CIB nmbr:* decimal digit, with a maximum value of 255 or register (1), (2) - (12).

The parameters are explained as follows:

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 is freed.

,BLOCK = *CIB addr*
specifies the address of the CIB 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.

Notes:

1. *When using any address returned from the EXTRACT macro instruction as input to the QEDIT macro instruction, the user must use the IEZCOM mapping macro to establish addressability based on the address returned by EXTRACT.*

2. *The CIB must reside in 24-bit addressable storage.*

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

Hexadecimal Code	Meaning
00	The required function was successfully completed.
04	The CIB to be deleted was not found on any CIB chain.
08	The limit for the number of CIBs to be chained was exceeded; an issuer who made a request to free all the CIBs on a chain was not in supervisor state and protect key zero; or the user provided an invalid address for the pointer to the CIB chain, an invalid address for the CIB address, or an invalid CIB number as input to the macro.

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)
```

RACDEF - Define a Resource to RACF

The RACDEF macro instruction is used to define, modify, or delete resource profiles for the Resource Access Control Facility (RACF). RACF uses the profiles to perform RACHECK authorization checking. RACHECK authorization checking verifies the user's authority to perform the corresponding resource manager function on the resource. The RACDEF caller must be authorized (APF-authorized, in system key 0-7, or in supervisor state).

A RACF user can change or add the RACDEF parameters, OWNER, LEVEL, UACC, or AUDIT by means of the RACDEF preprocessing and postprocessing exit routines. These routines are described in *System Programming Library: Resource Access Control Facility (RACF)*.

Systems using RACF Version 1, Release 6 or later, do not have to unconditionally deny all access requests that do not have sufficient authority. This release of RACF provides the option, through the specification of the WARNING parameter on the RACDEF macro, of issuing a warning message during RACHECK macro processing instead of failing the RACHECK request.

Note: Only callers in 24-bit addressing mode can issue this macro. Callers executing in 31-bit addressing mode, who want to utilize the RACDEF function, can code the RACROUTE macro.

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

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

ENTITY=*profile name addr*	*profile name addr:* A-type address, or register (2) - (12).
,VOLSER=*vol addr*	*vol addr:* A-type address, or register (2) - (12). **Note:** VOLSER is required only for CLASS='DATASET' and DSTYPE not equal to M when a discrete profile name is used.
,TYPE=DEFINE	
,TYPE=DEFINE,NEWNAME=*new dsn addr*	*new dsn addr:* A-type address, or register (2) - (12).
,TYPE=ADDVOL,OLDVOL=*old vol addr*	*old vol addr:* A-type address, or register (2) - (12).
,TYPE=CHGVOL,OLDVOL=*old vol addr*	
,TYPE=DELETE	**Default:** TYPE=DEFINE
,DSTYPE=N ,DSTYPE=V ,DSTYPE=M ,DSTYPE=T	**Default:** DSTYPE=N

,INSTLN = *parm list addr*	*parm list addr:* A-type address, or register (2) - (12).
,CLASS = *'classname'* ,CLASS = *class name addr*	*'classname':* 1-8 character name. *class name addr:* A-type address, or register (2) - (12). **Default:** CLASS = 'DATASET'
,MENTITY = *entity addr*	*entity addr:* A-type address, or register (2) - (12).
,MCLASS = *'classname'* ,MCLASS = *class name addr*	*'classname':* 1-8 character name. *class name addr:* A-type address, or register (2) - (12). **Default:** MCLASS = 'DATASET'
,MVOLSER = *volser addr*	*volser addr:* A-type address, or register (2) - (12).
,MGENER = ASIS ,MGENER = YES	**Default:** MGENER = ASIS
,ACEE = *acee addr*	*acee addr:* A-type address, or register (2) - (12).
,UNIT = *unit addr*	*unit addr:* A-type address, or register (2) - (12).
,SPECIAL = YES ,SPECIAL = NO	**Default:** SPECIAL = NO
,OWNER = *owner id addr*	*owner id addr:* A-type address, or register (2) - (12).
,LEVEL = *number* ,LEVEL = *reg*	**Default:** zero. *reg:* register (2) - (12).
,UACC = ALTER ,UACC = CONTROL ,UACC = UPDATE ,UACC = READ ,UACC = NONE ,UACC = *reg*	*reg:* register (2) - (12).
,DATA = *data addr*	*data addr:* A-type address or register (2) - (12).
,AUDIT = NONE ,AUDIT = *audit value* ,AUDIT = *(audit value (access level),audit value(access level),. . .)* ,AUDIT = *(reg)*	**Note:** AUDIT is valid only if TYPE = DEFINE is specified. *audit value:* ALL, SUCCESS, or FAILURES *access level:* READ, UPDATE, CONTROL, or ALTER **Default:** READ *reg:* register (2) - (12).
,RACFIND = YES ,RACFIND = NO	
,CHKAUTH = YES ,CHKAUTH = NO	**Default:** CHKAUTH = NO
,GENERIC = YES ,GENERIC = ASIS	**Default:** GENERIC = ASIS
,WARNING = YES ,WARNING = NO	**Default:** WARNING = NO **Note:** WARNING is valid only if TYPE = DEFINE is specified.
,RELEASE = *number*	*number:* 1.6 or 1.7 **Default:** RELEASE = 1.6
,FILESEQ = *reg* ,FILESEQ = *number*	*reg:* register (2) - (12). *number:* 1-9999
,EXPDT = *expir-date addr* ,RETPD = *retn-period addr*	*expir-date addr:* A-type address or register (2) - (12). *retn-period addr:* A-type address or register (2) - (12). **Default:** see description of parameter.
,ACCLVL = *(access value addr)* ,ACCLVL = *(access value addr, parm list addr)*	*access value addr:* A-type address or register (2) - (12). *parm list addr:* A-type address, or register (2) - (12).
,TAPELBL = STD ,TAPELBL = BLP ,TAPELBL = NL	**Default:** TAPELBL = STD
,CATGORY = *addr*	*addr:* A-type address, or register (2) - (12).
,SECLVL = *addr*	*addr:* A-type address, or register (2) - (12).
,ERASE = YES ,ERASE = NO	**Default:** ERASE = NO
,NOTIFY = *notify-id addr*	*notify-id addr:* A-type address or register (2) - (12).

The parameters are explained as follows:

ENTITY = *profile name addr*
specifies the address of the name of the discrete or generic profile that is to be defined to, modified, or deleted from RACF. The profile name is a 44-byte DASD data set name for CLASS='DATASET' or a 6-byte volume serial name for CLASS='DASDVOL' or CLASS='TAPEVOL'. The lengths of all other profile names are determined by the class descriptor table. The name must be left justified in the field and padded with blanks.

,VOLSER = *vol addr*
specifies the address of the volume serial number:

- For TYPE=ADDVOL, of the new volume to be added to the definition of the data set.
- For TYPE=ADDVOL and CLASS='TAPEVOL', of the new volume being added to the tape volume set identified by ENTITY.
- For TYPE=DEFINE and CLASS='DATASET', of the catalog (for a VSAM data set), or of the volume on which the data set resides (for a non-VSAM data set).

The volume serial number is optional if DSTYPE=M is specified; it is ignored if the profile name is generic.

The field pointed to by the specified address contains the volume serial number (padded to the right with blanks, if necessary, to make six characters).

,TYPE=DEFINE
,TYPE=DEFINE ,NEWNAME = *new dsn addr*
,TYPE=ADDVOL ,OLDVOL = *old vol addr*
,TYPE=CHGVOL ,OLDVOL = *old vol addr*
,TYPE=DELETE
specifies the type of action to be taken:

- TYPE=DEFINE - The definition of the resource is added to the RACF data set, and the current user is established as the owner of the defined entity.
- TYPE=DEFINE,NEWNAME= - If NEWNAME is specified, the address points to a 44-byte field containing the new name for the data set that is to be renamed. NEWNAME is only valid with CLASS= 'DATASET'. NEWNAME is not valid with DSTYPE=T.
- TYPE=ADDVOL - The new volume is added to the definition of the specified resource. For the DATASET class, the OLDVOL address specifies a previous volume of a multi-volume data set. For the TAPEVOL class, the ENTITY address specifies a previous volume of a tape volume set. This parameter applies only to discrete profiles.
- TYPE=CHGVOL - The volume serial number in the definition of the specified resource is changed from the old volume serial number identified in OLDVOL to the new volume serial number identified in the VOLSER parameter. This parameter applies only to discrete profiles. TYPE=CHGVOL is not valid with DSTYPE=T.

- TYPE=DELETE - The definition of the resource is removed from the RACF data set. (For a multivolume data set or a tape volume set, only the specified volume is removed from the definition.)

,DSTYPE=N
,DSTYPE=V
,DSTYPE=M
,DSTYPE=T
specifies the type of data set associated with the request:

- N for non-VSAM
- V for VSAM
- M for model profile
- T for tape

If DSTYPE=T is specified and tape data set protection is not active, the processing will be the same as for RACDEF CLASS='TAPEVOL'. Specify DSTYPE only for CLASS='DATASET'.

Note: Do not specify DSTYPE=M unless RACF Version 1, Release 4 or later is installed on your system.

,INSTLN=*parm list addr*
specifies the address of an area that is to contain parameter information meaningful to the RACDEF installation exit routines. This information is passed to the installation exit routines when they are given control from the RACDEF routine.

The INSTLN parameter can be used by an application program acting as a resource manager that needs to pass information to the RACDEF installation exit routines.

,CLASS=*'classname'*
,CLASS=*class name addr*
specifies that a profile is to be defined, modified, or deleted in the specified class. If an address is specified, the address must point to a one-byte length field followed by the class name (for example, DATASET or TAPEVOL). The class name should be no longer than eight characters.

,MENTITY=*entity addr*
specifies the address of the name of the discrete or generic profile profile that is to be used as a model in defining the ENTITY profile. The profile can belong to any class, as specified by the MCLASS parameter, and can be either a discrete or a generic profile. MENTITY can be specified with TYPE=DEFINE but not with TYPE=DEFINE,NEWNAME=*new dsn addr*. The name is contained in a 44-byte field pointed to by the specified address. The name is left justified in the field and padded with blanks.

,MCLASS=*'classname'*
,MCLASS=*class name addr*
specifies the class to which the profile defined by MENTITY= belongs. If an address is specified, the address must point to a one-byte length field followed by the class name. The class name should be no longer than eight characters. The default is MCLASS='DATASET'.

,MVOLSER = *volser addr*

specifies the address of the volume serial number of the volume associated with the profile in the MENTITY operand. The field pointed to by the specified address contains the volume serial number, padded to the right with blanks, if necessary, to make six characters.

This parameter is required if MENTITY specifies a discrete profile name in the DATASET class.

,MGENER = ASIS
,MGENER = YES

specifies whether the profile name defined by MENTITY is to be treated as a generic name.

- If MGENER = ASIS is specified, the profile name is considered a generic only if it contains a generic character: an asterisk (*) or a percent sign (%).

- If MGENER = YES is specified, the profile name is considered a generic, even if it does not contain a generic character: an asterisk (*) or a percent sign (%).

MGENER is ignored if the GENCMD option on the RACF SETROPTS command is not specified for the class (see *RACF Command Language Reference*).

,ACEE = *acee addr*

specifies the address of the accessor environment element (ACEE) to be used during RACDEF processing. If no ACEE is specified, RACF uses the TASK ACEE pointer (TCBSENV) in the extended TCB. If the TASK ACEE pointer is zero, RACF uses the main ACEE. The main ACEE's address is in the ASXBSENV field in the address space extension block.

,UNIT = *unit addr*

specifies the address of a field containing unit information. UNIT is valid only if TYPE = CHGVOL or TYPE = DEFINE is specified. If a unit address is specified, the unit information in the data set profile is replaced by the unit information pointed to by this unit address. The unit address must point to a field containing a one-byte length field (whose value can range from 4 through 8) followed by the actual unit information. If the value in the length field is 4, the unit information is assumed to contain a copy of the information in the UCBTYP field of the UCB. Otherwise the unit information is assumed to be in the generic form (for example, 3330-1). This parameter is ignored for generic names.

,SPECIAL = YES
,SPECIAL = NO

specifies whether or not a RACDEF operation is to be completed if the requestor has the SPECIAL attribute.

This keyword is designated primarily for use by RACF commands.

,OWNER = *owner id addr*

specifies the address of a field containing the profile owner's id. OWNER is valid if TYPE = DEFINE is specified. The owner's id must be a valid (RACF-defined) userid or group name. The address must point to an 8-byte field containing the owner's name, left-justified and padded with blanks.

Note: RACF does not check the validity of the owner's id if it has been added or modified by the RACDEF preprocessing and/or postprocessing exit routines.

,LEVEL = *number*
,LEVEL = *reg*

specifies a level value for the profile. LEVEL is valid only if TYPE = DEFINE is specified. The level number must be a valid decimal number in the range 0 to 99. If a register is specified, its low-order byte must contain the binary representation of the number.

Note: RACF does not check the validity of this number if it has been added or modified by the RACDEF preprocessing and/or postprocessing exit routines.

,UACC = ALTER
,UACC = CONTROL
,UACC = UPDATE
,UACC = READ
,UACC = NONE
,UACC = *reg*

specifies a universal access authority for the profile. UACC is valid only if TYPE = DEFINE is specified. UACC must contain a valid access authority (ALTER, CONTROL, UPDATE, READ, or NONE). If a register is specified, the low-order byte must contain one of the following valid access authorities:

X'80' - ALTER
X'40' - CONTROL
X'20' - UPDATE
X'10' - READ
X'01' - NONE

Note: RACF does not check the validity of the universal access authority if it has been added or modified by the RACDEF preprocessing and/or postprocessing exit routine.

,DATA = *data addr*

specifies the address of a field that contains up to 255 characters of installation-defined data to be placed in the profile. DATA is valid only if TYPE = DEFINE is specified. The data address must point to a field containing a one-byte length field (whose value can range from 0 to 255) followed by the actual installation-defined data.

,AUDIT = NONE
,AUDIT = *audit value*
,AUDIT = *(audit value(access level),audit value(access level),. . .)*
,AUDIT = *(reg)*

specifies the types of accesses and the access levels that are to be logged to the SMF data set. AUDIT is valid only if TYPE = DEFINE is specified.

For *audit value*, specify one of the following: ALL, SUCCESS, or FAILURES. You may optionally specify an *access level*(access authority) following each *audit value*.

Access Levels:

- READ, the default access level value, logs access attempts at any level.
- UPDATE logs access attempts at the UPDATE, CONTROL, and ALTER levels.
- CONTROL logs access attempts at the CONTROL and ALTER levels.
- ALTER logs access attempts at the ALTER level only.

Note: For more information about specific audit values and access levels, please see the *RACF Command Language Reference*.

RACF resolves combinations of conflicting specifications by using the most encompassing specification. Thus, in the case of the following:

```
ALL(UPDATE),FAILURES(READ)
```

RACF assumes SUCCESS(UPDATE),FAILURES(READ).

For compatibility with previous releases, register notation can also be specified as AUDIT=*reg* if the register is not given as a symbolic name ALL, SUCCESS, or FAILURES.

Logging is controlled separately for SUCCESS and FAILURES, and can also be suppressed or requested via the RACHECK post-processing installation exit routine.

If a register is specified, its low-order byte must contain one of the following valid audit values:

Bit	Meaning
0	ALL
1	SUCCESS
2	FAILURES
3	NONE
4-5	Qualifier for SUCCESS
6-7	Qualifier for FAILURES

The qualifier codes are as follows:

00	READ
01	UPDATE
10	CONTROL
11	ALTER

Only one of bits 0-3 can be on. If ALL is specified, the two qualifier fields can be used to request different logging levels for successful and unsuccessful events.

Note: RACF does not check the validity of the audit type if it has been added or modified by the RACDEF preprocessing and/or postprocessing exit routine.

,RACFIND=YES
,RACFIND =NO
specifies whether or not a discrete profile is involved in RACDEF processing. When TYPE=DEFINE is specified, RACFIND=YES means that a discrete profile is to be created. When TYPE=DELETE, DEFINE with NEWNAME, CHGVOL, or ADDVOL is specified, RACFIND=YES means that a discrete profile already exists.

RACFIND=NO means (when TYPE=DEFINE) that no discrete profile is to be created, but some authorization checking is required. For other types of action, no discrete profile should exist.

,CHKAUTH=YES
,CHKAUTH=NO

specifies whether or not an internal RACHECK ATTR=ALTER is to be done to verify that the user is authorized to perform the operation.

CHKAUTH=YES is valid when either TYPE=DEFINE,NEWNAME= or TYPE=DELETE is specified.

For DSTYPE=T, specifies that an internal RACHECK ATTR=UPDATE will be done to verify that the user is authorized to define a data set (TYPE=DEFINE), delete a data set (TYPE=DELETE), or add a volume (TYPE=ADDVOL).

,RELEASE=*number*

specifies the RACF release level of the parameter list to be generated by this macro.

Certain parameters can be specified only with particular releases. For instance, to use the RACFDEF release 1.7 parameter EXPDT you must be using RACF1.7 on your system and specify RELEASE=1.7. If you specify a parameter with an incompatible release level, the parameter will not be accepted by the macro processing. An error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 7 on page 2-276.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACDEF macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

,FILESEQ=*number*
,FILESEQ=*reg*

specifies the file sequence number of a tape data set on a tape volume or within a tape volume set. The *number* must be in the range 1 - 9999. If a register is specified, it must contain the file sequence number in the low-order half-word. If CLASS='DATASET' and DSTYPE=T are not specified, FILESEQ is ignored.

,EXPDT=*expir-date addr*
,RETPD=*retn-period addr*

specifies the address containing information about the data set's expiration date or RACF security retention period.

EXPDT=*expir-date addr* specifies the address of a three-byte field containing the data set's expiration date. The date is given in packed decimal form as YYDDDF, where YY is the year and DDD is the day number. The year must be in the range 01 through 99, and the day number must be in the range 1 through 366.

RETPD=*retn-period addr* specifies the address of a two-byte field containing the number of days after which RACF protection for the data set expires. The value specified must be in the range 1 through 9999.

If neither EXPDT nor RETPD is specified, a default RACF security retention period is obtained from the RETPD keyword specified on a prior RACF SETROPTS command.

These parameters are valid only if CLASS='DATASET' and DSTYPE=T.

,ACCLVL=*(access value addr)*
,ACCLVL=(*access value addr,parm list addr***)**
specifies the tape label access level information for the MVS tape label functions. The address must point to a field containing a one-byte length field (with a value that can range from 0-8) followed by an eight-character string that will be passed to the RACDEF installation exit routines. The parameter list address points to a parameter list containing additional information to be passed to the RACDEF installation exit routines.

RACF does not check or modify this information.

,TAPELBL=STD
,TAPELBL=BLP
,TAPELBL=NL
specifies the type of tape labelling to be done:

- STD - IBM or ANSI standard labels.
- BLP - bypass label processing.
- NL - non-labeled tapes.

For TAPELBL=BLP, the user must have the requested authority to the profile ICHBLP in the general resource class FACILITY. For TAPELBL=NL or BLP, the user will not be allowed to protect volumes with volume serial numbers in the format "Lnnnnn."

The TAPELBL parameter is passed to the RACDEF installation exits.

This parameter is primarily intended for use by data management routines to indicate the label type from the LABEL keyword on the JCL statement.

This parameter is only valid for CLASS='DATASET' and DSTYPE=T, or CLASS='TAPEVOL'. The default is TAPELBL=STD

,CATGORY=*addr*
Specifies the address of a list of installation-defined category name identifiers. Each identifier is a two-byte field whose binary value identifies an entry in the installation-defined list of category names.

The list starts with a full word that contains the number of entries in the list.

This keyword is designated primarily for use by RACF commands.

,SECLVL=*addr*
Specifies the address of a list of installation-defined security level identifiers. Each identifier is a half word, containing a value that corresponds to an installation-defined security level name.

The identifiers must be in the range 1 - 254. Only one identifier may be passed in the list.

The list must start with a full word containing the number of entries in the list (currently, only 0 or 1).

,ERASE=YES
,ERASE=NO

specifies whether the DASD data set, or the released space, is to be erased when it is deleted or part of its space is to be released for reuse.

- If ERASE=YES is specified, the data set will be erased when it is deleted, or released for reuse.
- If ERASE=NO is specified, the data set will not be erased, deleted, or released.

Note: This parameter may be overridden by the RACF SETROPTS command.

The default is ERASE=NO.

,NOTIFY=*notify-id addr*

specifies the address of an eight-byte area containing the userid of the RACF-defined user who is to be notified when an unauthorized attempt to access the resource protected by this profile is detected.

,GENERIC=YES
,GENERIC=ASIS

specifies whether the resource name is treated as a generic profile name. If GENERIC is specified with CLASS=DEFINE, NEWNAME, then GENERIC applies to both the old and new names. GENERIC is ignored if the GENCMD option on the RACF SETROPTS command is not specified for the class (see *RACF Command Language Reference*).

This keyword is designed primarily for use by RACF commands.

- If GENERIC=YES is specified, the resource name is considered a generic profile name, even if it does not contain a generic character: an asterisk (*) or a percent sign (%).
- If GENERIC=ASIS is specified, the resource name is considered a generic only if it contains a generic character: an asterisk (*) or a percent sign (%).

,WARNING=YES
,WARNING =NO

WARNING is valid only if TYPE=DEFINE is specified. If WARNING=YES is specified, access is granted to the resource and the event is logged as a warning if either the SUCCESS and/or FAILURES logging is requested.

This keyword is designed primarily for use by RACF commands.

Parameters For RELEASE = 1.6 and Later

The RELEASE values for which a parameter is valid are marked with an 'X'.

Parameter	RELEASE = 1.6	RELEASE = 1.7
ACEE =	X	X
ACCLVL =		X
AUDIT =	X	X
CATEGORY =		X
CHKAUTH =	X	X
CLASS =	X	X
DATA =	X	X
DSTYPE = N, V, or M	X	X
DSTYPE = T		X
ENTITY =	X	X
ERASE =		X
EXPDT =		X
FILESEQ =		X
GENERIC =	X	X
INSTLN =	X	X
LEVEL =	X	X
MCLASS =		X
MENTITY =	X	X
MGENER =		X
MVOLSER =	X	X
NOTIFY =		X
OWNER =	X	X
RACFIND =	X	X
RELEASE =	X	X
RETPD =		X
SECLVL =		X
TAPELBL =		X
TYPE =	X	X
SPECIAL =	X	X
UACC =	X	X
UNIT =	X	X
VOLSER =	X	X
WARNING =	X	X

Figure 7. RACDEF Parameters for RELEASE = 1.6 and Later

Return Codes and Reason Codes

When control is returned, register 15 contains one of these return codes. (If register 15 contains 0, then register 0 contains a reason code.)

Hexadecimal Code	Meaning
00	RACDEF has completed successfully. Register 0 contains one of the following reason codes: 00 indicates a normal completion. 04 indicates RACFIND=NO was specified and no generic profile applying to the data set was found.
04	RACDEF has completed processing. Register 0 contains one of the following reason codes: 00 indicates the following: - For TYPE=DEFINE, the resource name was previously defined. - For TYPE=DEFINE,NEWNAME, the new resource name was previously defined. - For TYPE=DELETE, the resource name was not previously defined. 04 indicates for TYPE=DEFINE that the dataset name was previously defined on a different volume and that the option disallowing duplicate datasets was specified in ICHSECOP at IPL.
08	RACDEF has completed processing. Register 0 contains one of the following reason codes: 00 indicates the following: - For TYPE=DEFINE, the check for authority to allocate a data set or create a profile with the specified name has been failed. - For TYPE=DELETE or TYPE=DEFINE,NEWNAME if CHKAUTH=YES is specified, the authorization check has been failed. - For TYPE=ADDVOL,OLDVOL the old value was not defined. 04 indicates for TYPE=DEFINE that no profile was found to protect the dataset and that the RACF protect-all option is in effect. 08 indicates TYPE=DEFINE or TYPE=ADDVOL,OLDVOL and DSTYPE=T were specified. And the user is not authorized to define a data set on the specified volume. 0C indicates TYPE=DEFINE and DSTYPE=T were specified. And the user is not authorized to define a data set with the specified name. 10 indicates DSTYPE=T or CLASS=TAPEVOL was specified. And the user is not authorized to specify LABEL=(,BLP).

Hexadecimal Code	Meaning
0C	For TYPE=DEFINE,NEWNAME, the old data set name was not defined; or if the generation data group (GDG) modeling function is active, an attempt was made to rename a GDG name to a name that requires the creation of a new profile; or if generic profile checking is active, the old data set name was protected by a generic profile and there is no generic profile that will protect the new data set name. This last case refers only to an attempt to rename an existing profile, which cannot be found.
10	For TYPE=DEFINE with MENTITY, the model resource was not defined.
64	Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACDEF macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

Example

Operation: Invoke RACF to define a non-VSAM data set residing on the volume pointed to by register 8. Register 7 points to the data set name. All successful requests for update authority to the data set are to be audited, as well as all unsuccessful ones. A discrete profile is to be created.

```
RACDEF  ENTITY=(R7),VOLSER=(R8),CLASS='DATASET',
        AUDIT=(SUCCESS(UPDATE),FAILURES),
        RACFIND=YES
```

RACDEF (List Form)

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

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

ENTITY=*profile name addr*	*profile name addr:* A-type address. **Note:** ENTITY must be specified on either the list or the execute form of the macro.
,VOLSER=*vol addr*	*vol addr:* A-type address, or register (2) - (12). **Note:** VOLSER is required only for CLASS='DATASET' and DSTYPE not equal to M when a discrete profile name is used.
,TYPE=DEFINE ,TYPE=DEFINE,NEWNAME=*new dsn addr* ,TYPE=ADDVOL,OLDVOL=*old vol addr* ,TYPE=CHGVOL,OLDVOL=*old vol addr* ,TYPE=DELETE	*new dsn addr:* A-type address, or register (2) - (12). *old vol addr:* A-type address, or register (2) - (12). **Default:** TYPE=DEFINE
,DSTYPE=N ,DSTYPE=V ,DSTYPE=M ,DSTYPE=T	**Default:** DSTYPE=N
,INSTLN=*parm list addr*	*parm list addr:* A-type address, or register (2) - (12).
,CLASS='*classname*' ,CLASS=*class name addr*	*'classname':* 1-8 character name. *class name addr:* A-type address, or register (2) - (12). **Default:** CLASS='DATASET'
,MENTITY=*entity addr*	*entity addr:* A-type address, or register (2) - (12).
,MCLASS='*classname*' ,MCLASS=*class name addr*	*'classname':* 1-8 character name. *class name addr:* A-type address, or register (2) - (12). **Default:** MCLASS='DATASET'
,MVOLSER=*volser addr*	*volser addr:* A-type address, or register (2) - (12).
,MGENER=ASIS ,MGENER=YES	**Default:** MGENER=ASIS
,ACEE=*acee addr*	*acee addr:* A-type address, or register (2) - (12).
,UNIT=*unit addr*	*unit addr:* A-type address, or register (2) - (12).
,SPECIAL=YES ,SPECIAL=NO	**Default:** SPECIAL=NO
,OWNER=*owner id addr*	*owner id addr:* A-type address, or register (2) - (12).
,LEVEL=*number* ,LEVEL=*reg*	**Default:** zero. *reg:* register (2) - (12).

,UACC=ALTER ,UACC=CONTROL ,UACC=UPDATE ,UACC=READ ,UACC=NONE ,UACC=*reg*	*reg:* register (2) - (12).
,DATA=*data addr*	*data addr:* A-type address or register (2) - (12).
,AUDIT=NONE ,AUDIT=*audit value* ,AUDIT=(*audit value (access level),audit value(access level)*) ,AUDIT=*(reg)*	*audit value:* ALL, SUCCESS, or FAILURES *access level:* READ, UPDATE, CONTROL, or ALTER **Default:** READ *reg:* register (2) - (12).
,RACFIND=YES ,RACFIND=NO	
,CHKAUTH=YES ,CHKAUTH=NO	**Default:** CHKAUTH=NO
,GENERIC=YES ,GENERIC=ASIS	**Default:** GENERIC=ASIS
,WARNING=YES ,WARNING=NO	**Default:** WARNING=NO **Note:** Warning is valid only if TYPE=DEFINE is specified.
,RELEASE=*number*	*number:* 1.6 or 1.7 **Default:** RELEASE=1.6
,FILESEQ=*reg* ,FILESEQ=*number*	*reg:* register (2) - (12). *number:* 1-9999
,EXPDT=*expir-date addr* ,RETPD=*retn-period addr*	*expir-date addr:* A-type address or register (2) - (12). *retn-period addr:* A-type address or register (2) - (12). **Default:** see description of parameter.
,ACCLVL=*(access value addr)* ,ACCLVL=(*access value addr, parm list addr)*	*access value addr:* A-type address or register (2) - (12). *parm list addr:* A-type address, or register (2) - (12).
,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default:** TAPELBL=STD
,CATGORY=*addr*	*addr:* A-type address, or register (2) - (12).
,SECLVL=*addr*	*addr:* A-type address, or register (2) - (12).
,ERASE=YES ,ERASE=NO	**Default:** ERASE=NO
,NOTIFY=*notify-id addr*	*notify-id addr:* A-type address or register (2) - (12).
,MF=L	

The parameters are explained under the standard form of the RACDEF macro instruction, with the following exception:

,MF=L
specifies the list form of the RACDEF macro instruction.

RACDEF (Execute Form)

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

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

ENTITY=*profile name addr*	*profile name addr:* RX-type address. **Note:** ENTITY must be specified on either the list or the execute form of the macro.
,VOLSER=*vol addr*	*vol addr:* RX-type address, or register (2) - (12). **Note:** VOLSER is required only for CLASS='DATASET' and DSTYPE not equal to M when a discrete profile name is used.
,TYPE=DEFINE ,TYPE=DEFINE,NEWNAME=*new dsn addr* ,TYPE=ADDVOL,OLDVOL=*old vol addr* ,TYPE=CHGVOL,OLDVOL=*old vol addr* ,TYPE=DELETE	*new dsn addr:* RX-type address, or register (2) - (12). *old vol addr:* RX-type address, or register (2) - (12). **Default:** TYPE=DEFINE
,DSTYPE=N ,DSTYPE=V ,DSTYPE=M ,DSTYPE=T	**Default:** DSTYPE=N
,INSTLN=*parm list addr*	*parm list addr:* RX-type address, or register (2) - (12).
,CLASS='*classname*' ,CLASS=*class name addr*	*'classname':* 1-8 character name. *class name addr:* RX-type address, or register (2) - (12). **Default:** CLASS='DATASET'
,MENTITY=*entity addr*	*entity addr:* RX-type address, or register (2) - (12).
,MCLASS='*classname*' ,MCLASS=*class name addr*	*'classname':* 1-8 character name. *class name addr:* RX-type address, or register (2) - (12). **Default:** MCLASS='DATASET'
,MVOLSER=*volser addr*	*volser addr:* RX-type address, or register (2) - (12).
,MGENER=ASIS ,MGENER=YES	**Default:** MGENER=ASIS
,ACEE=*acee addr*	*acee addr:* RX-type address, or register (2) - (12).
,UNIT=*unit addr*	*unit addr:* RX-type address, or register (2) - (12).
,SPECIAL=YES ,SPECIAL=NO	**Default:** SPECIAL=NO
,OWNER=*owner id addr*	*owner id addr:* RX-type address, or register (2) - (12).
,LEVEL=*number* ,LEVEL=*reg*	**Default:** zero. *reg:* register (2) - (12).

,UACC=ALTER ,UACC=CONTROL ,UACC=UPDATE ,UACC=READ ,UACC=NONE ,UACC=*reg*	*reg:* register (2) - (12).
,DATA=*data addr*	*data addr:* RX-type address or register (2) - (12).
,AUDIT=NONE ,AUDIT=*audit value* ,AUDIT=(*audit value (access level),audit value(access level)*) ,AUDIT=*(reg)*	*audit value:* ALL, SUCCESS, or FAILURES *access level:* READ, UPDATE, CONTROL, or ALTER **Default:** READ *reg:* register (2) - (12).
,RACFIND=YES ,RACFIND=NO	
,CHKAUTH=YES ,CHKAUTH=NO	**Default:** CHKAUTH=NO
,GENERIC=YES ,GENERIC=ASIS	**Default:** GENERIC=ASIS
,WARNING=YES ,WARNING=NO	**Default:** WARNING=NO **Note:** Warning is valid only if TYPE=DEFINE is specified.
,RELEASE=*(number*,CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number:* 1.6 or 1.7 **Default:** RELEASE=1.6
,FILESEQ=*reg* ,FILESEQ=*number*	*reg:* register (2) - (12). *number:* 1-9999
,EXPDT=*expir-date addr* ,RETPD=*retn-period addr*	*expir-date addr:* RX-type address or register (2) - (12). *retn-period addr:* RX-type address or register (2) - (12). **Default:** see description of parameter.
,ACCLVL=*(access value addr)* ,ACCLVL=(*access value addr, parm list addr)*	*access value addr:* RX-type address or register (2) - (12). *parm list addr:* A-type address, or register (2) - (12).
,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default:** TAPELBL=STD
,CATGORY=*addr*	*addr:* RX-type address, or register (2) - (12).
,SECLVL=*addr*	*addr:* RX-type address, or register (2) - (12).
,ERASE=YES ,ERASE=NO	**Default:** ERASE=NO
,NOTIFY=*notify-id addr*	*notify-id 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 RACDEF macro instruction, with the following exception:

,MF=(E,*ctrl addr***)**
specifies the execute form of the RACDEF macro instruction using a remote control program parameter list.

,RELEASE=(*number***,CHECK)**
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

Certain parameters can be specified only with particular releases. For instance, to use the RACDEF release 1.7 parameter EXPDT you must be using RACF 1.7 on your system and specify RELEASE=1.7. If you specify a parameter with an incompatible release level, the parameter will not be accepted by the macro processing. An error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 7 on page 2-276.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACDEF macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done. Instead, a return code X'64' will be generated.

RACHECK - Check RACF Authorization

The RACHECK macro instruction is used to provide authorization checking when a user requests access to a RACF-protected resource. The RACHECK macro instruction is also described in the *Supervisor Services and Macro Instructions,* with the exception of the PROFILE, CSA, ACEE, and LOG parameters. These parameters are restricted in use and must only be used by programs that are authorized (APF-authorized, in system key 0-7, or in supervisor state).

Systems using RACF Version 1, Release 6 or later, have the option to temporarily grant access requests to users who do not have sufficient authority instead of unconditionally denying requests. In this case, RACF issues a warning message instead of failing the request. RACF provides this option on an individual basis; installations can use the warning facility selectively via the WARNING=YES keyword of the RACDEF macro for that particular profile, without affecting the access control provided by other RACF profiles.

RACHECK bypasses the warning processing if the OWNER keyword is specified, as this indicates that the request is coming from a RACF command processor.

Notes:

1. *Do not use the DSTYPE=M or OWNER parameters unless RACF Version 1, Release 4 or later is installed on your system.*

2. *Do not use the ACCLVL or RACFIND parameters unless RACF Version 1, Release 5 or later is installed on your system.*

3. *Only callers in 24-bit addressing mode can issue this macro. Callers executing in 31-bit addressing mode, who want to use the RACHECK function, can code the RACROUTE macro.*

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

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

PROFILE=*profile addr* ENTITY=(*resource name addr*) ENTITY=(*resource name addr,CSA*)	*profile addr:* A-type address, or register (2) - (12). *resource name addr:* A-type address, or register (2) - (12).
,VOLSER=*vol addr*	*vol addr:* A-type address, or register (2) - (12). **Note:** VOLSER is required only for CLASS='DATASET' and DSTYPE not equal to M when a discrete profile name is used and only when ENTITY is also coded.
,CLASS='*classname*' ,CLASS=*class name addr*	*'classname':* 1-8 character name. *class name addr:* A-type address, or register (2) - (12). **Default:** CLASS='DATASET'
,RELEASE=*number*	*number:* 1.6 or 1.7 **Default:** RELEASE=1.6
,ATTR=READ ,ATTR=UPDATE ,ATTR=CONTROL ,ATTR=ALTER ,ATTR=*reg*	*reg:* register (2) - (12). **Default:** ATTR=READ
,DSTYPE=N ,DSTYPE=V ,DSTYPE=M ,DSTYPE=T	**Default:** DSTYPE=N
,INSTLN=*parm list addr*	*parm list addr:* A-type address, or register (2) - (12).
,LOG=ASIS ,LOG=NOFAIL ,LOG=NONE ,LOG=NOSTAT	**Default:** LOG=ASIS
,OLDVOL=*old vol addr*	*old vol addr:* A-type address, or register (2) - (12).
,APPL='*applname*' ,APPL=*applname addr*	*applname addr:* A-type address, or register (2) - (12).
,ACEE=*acee addr*	*acee addr:* A-type address, or register (2) - (12).
,OWNER=*owner ID addr*	*owner ID addr:* A-type address, or register (2) - (12).
,ACCLVL=(*access value addr*) ,ACCLVL=(*access value addr,* *parm list addr*)	*access value addr:* A-type address or register (2)-(12). *parm list addr:* A-type address or register (2)-(12).
,RACFIND=YES ,RACFIND=NO	
,GENERIC=YES ,GENERIC=ASIS	**Default:** GENERIC=ASIS
,FILESEQ=*reg* ,FILESEQ=*number*	*reg:* register (2) - (12). *number:* 1-9999
,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default:** TAPELBL=STD
,STATUS=NONE ,STATUS=ERASE	**Default:** STATUS=NONE

The parameters are explained as follows:

PROFILE=*profile addr*
ENTITY=*(resource name addr)*
ENTITY=*(resource name addr, CSA)*

ENTITY=*(resource name addr)* specifies that RACF authorization checking is to be performed for the resource whose name is pointed to by the specified address. The resource name is a 44-byte DASD data set name for CLASS='DATASET' or a 6-byte volume serial number for CLASS='DASDVOL' or CLASS='TAPEVOL'. The length of all other resource names is determined from the class descriptor tables. The name must be left justified in the field and padded with blanks.

PROFILE=*profile addr* specifies that RACF authorization checking is to be performed for the resource whose profile is pointed to by the specified address.

ENTITY=*(resource name addr,CSA)* specifies that RACF authorization checking is to be performed for the indicated resource, and that a copy of the profile is to be maintained in main storage. The storage acquired for the profile is obtained from the common storage area (CSA), and is fetch-protected, key zero storage. The issuer of RACHECK must free this storage when the profile is no longer needed. (The profile subpool number and length are part of the profile data being returned.) If CSA is specified and the return code produced by the RACHECK macro instruction is 00 or 08, the address of the profile is returned in register 1.

By establishing and maintaining a resource profile, the resource manager can reduce the I/O required to perform RACF authorization checks on highly-accessed resources.

,VOLSER=*vol addr*

specifies the volume serial number, as follows:

- For non-VSAM DASD data sets and tape data sets, this is the volume serial number of the volume on which the data set resides.
- For VSAM DASD data sets and tape data sets, this is the volume serial number of the catalog controlling the data set.

The volume serial number is optional if DSTYPE=M is specified; it is ignored if the profile name is generic.

The field pointed to by the specified address contains the volume serial number padded to the right with blanks, if necessary, to make six characters. VOLSER= is only valid and must be supplied with CLASS='DATASET', (unless DSTYPE=M is specified) and if ENTITY is also coded.

,CLASS=*'classname'*
,CLASS=*classname addr*

specifies that RACF authorization checking is to be performed for a resource of the specified class. If an address is specified, the address must point to a one-byte field indicating the length of the classname, followed by the class name (for example DATASET, DASDVOL or TAPEVOL).

,RELEASE=*number*

specifies the RACF release level of the parameter list to be generated by this macro.

Certain parameters can be specified only with particular releases. For instance, to use the RACHECK release 1.7 parameter FILESEQ you must be using RACF 1.7 on your system and specify RELEASE=1.7. If you specify a parameter with an incompatible release level, the parameter will not be accepted by the macro processing. An error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 9 on page 2-292.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACHECK macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

,ATTR=READ
,ATTR=UPDATE
,ATTR=CONTROL
,ATTR=ALTER
,ATTR=*reg*

specifies the access authority of the user or group permitted access to the resource for which RACF authorization checking is to be performed:

READ - RACF user or group can open the resource only to read.

UPDATE - RACF user or group can open the resource to write or read.

CONTROL - For VSAM data sets, RACF user or group has authority equivalent to the VSAM control password. For non-VSAM data sets and other resources, RACF user or group has UPDATE authority.

ALTER - RACF user or group has total control over the resource.

If a register is specified, the register must contain one of the following codes in the low-order byte of the register:

X'02' - READ
X'04' - UPDATE
X'08' - CONTROL
X'80' - ALTER

,DSTYPE=N
,DSTYPE=V
,DSTYPE=M
,DSTYPE=T

specifies the type of data set associated with the request:

- N for non-VSAM
- V for VSAM
- M for model profile
- T for tape

If DSTYPE=T is specified and tape data set protection is not active, the processing will be the same as for RACHECK CLASS='TAPEVOL'. DSTYPE should only be specified for CLASS='DATASET'.

Note: Do not specify DSTYPE=M unless RACF Version 1, Release 4 or later is installed on your system.

,INSTLN=*parm list addr*
specifies the address of an area that is to contain parameter information meaningful to the RACHECK installation exit routine. This information is passed to the installation exit routine when it is given control from the RACHECK routine.

The INSTLN parameter can be used by an application program acting as a resource manager that needs to pass information to the RACHECK installation exit routine.

,LOG=ASIS
,LOG=NOFAIL
,LOG=NONE
,LOG=NOSTAT
specifies the types of access attempts to be recorded on the SMF data set:

ASIS - Attempts to be recorded are as specified on the ADDSD or ALTDSD command that was issued for the data set or the RDEFINE or RALTER command for the tape or DASD volume.

NOFAIL - If the authorization check fails, the attempt is not recorded. If the authorization check succeeds, the attempt is recorded as in ASIS.

NONE - The attempt is not to be recorded.

NOSTAT - The attempt is not to be recorded and no resource statistics are to be updated.

,OLDVOL=*old vol addr*
specifies a volume serial:

- For CLASS='DATASET', within the same multivolume data set specified by VOLSER=.
- For CLASS='TAPEVOL', within the same tape volume specified by ENTITY=.

RACF authorization checking will verify that the OLDVOL specified is part of the same multivolume data set or tape volume set.

The specified address points to the field that contains the volume serial number padded to the right with blanks, if necessary, to make six characters.

,APPL=*'applname'*
,APPL=*applname addr*
specifies the name of the application requesting authorization checking. The *applname* is not used for the authorization checking process but is made available to the installation exit routine(s) called by the RACHECK routine. If the address is specified, the address must point to an 8-byte field containing the application name left justified and padded with blanks.

,ACEE=*acee addr*

specifies the address of the accessor environment element (ACEE) to be used for checking authorization during RACHECK processing. If no ACEE is specified, RACF uses the TASK ACEE pointer (TCBSENV) in the extended TCB. Otherwise, or if the TASK ACEE pointer is zero, RACF uses the main ACEE for the address space. The main ACEE is pointed to by the ASXBSENV field of the address space extension block.

,OWNER=*owner ID addr*

specifies a profile owner id that is compared with the profile owner id in the owner field of the RACF profile. If the owner names match, the access authority allowed for that userid is 'ALTER'. The address must point to an 8-byte field containing the owner name, left-justified and padded with blanks.

If OWNER is specified, any WARNING and OPERATIONS attribute processing is bypassed.

This keyword is primarily intended for use by the RACF command processors to check a user's authority to access the profile specified.

Note: Do not specify OWNER unless RACF Version 1, Release 4 or later is installed on your system.

,ACCLVL=*(access value addr)*
,ACCLVL=*(access value addr,parm list addr)*

specifies the tape label access level information for the MVS tape label functions. The access value pointed to by the specified address is a one byte length field, containing the value (0-8) of the length of the following data, followed by an eight-character string that will be passed to the RACHECK installation exit routines. The optional parameter list pointed to by the specified address contains additional information to be passed to the RACHECK installation exit routines. RACF does not inspect or modify this information.

Note: Do not use the ACCLVL parameter unless RACF Version 1, Release 5 or later is installed on your system.

,RACFIND=YES
,RACFIND=NO

indicates whether or not the resource is protected by a discrete profile. The RACF processing and the possible return codes are given in Figure 8. Note that in all cases, a return code of X'0C' is also possible.

Note: Do not use the RACFIND parameter unless RACF Version 1, Release 5 or later is installed on your system.

Operand	Generic Profile Checking Inactive	Generic Profile Checking Active
RACFIND=YES	Look for discrete profile; if found, exit with return code 00 or 08. If no discrete profile is found, exit with return code 08.	Look for discrete profile; if found, exit with return code 00 or 08. Look for generic profile; if found, exit with return code 00 or 08. Exit with return code 08 if neither a discrete nor a generic profile is found.
RACFIND=NO	No checking. Exit with return code 04.	Look for generic profile; if found, exit with return code 00 or 08. if not found, exit with return code 04.
RACFIND not specified	Look for discrete profile; if found, exit with return code 00 or 08. If no discrete profile is found, exit with return code 04.	Look for discrete profile; if found, exit with return code 00 or 08. Look for generic profile; if found, exit with return code 00 or 08. Exit with return code 04 if neither a discrete nor a generic profile is found.

Figure 8. Types of Profile Checking Performed by RACHECK

,GENERIC=YES
,GENERIC=ASIS

specifies whether the resource name is to be treated as a generic profile name. If GENERIC is specified with CLASS=DEFINE, NEWNAME, then GENERIC applies to both the old and new names. GENERIC is ignored if the GENCMD option on the RACF SETROPTS command is not specified for the class (see *RACF Command Language Reference*).

This keyword is designed primarily for use by RACF commands.

- If GENERIC=YES is specified, the resource name is considered a generic profile name, even if it does not contain either of the generic characters: an asterisk (*) or a percent sign (%).
- If GENERIC=ASIS is specified, the resource name is considered a generic only if it contains either of the generic characters: an asterisk (*) or a percent sign (%).

,FILESEQ=*number*
,FILESEQ=*reg*

specifies the file sequence number of a tape data set on a tape volume or within a tape volume set. The value must be in the range 1 - 9999. If a register is specified, it must contain the file sequence number in the low-order half-word. If CLASS='DATASET' and DSTYPE=T are not specified, FILESEQ is ignored.

,TAPELBL=STD
,TAPELBL=BLP
,TAPELBL=NL

specifies the type of tape label processing to be done:

- STD - IBM or ANSI standard labels.
- BLP - bypass label processing.
- NL - non-labeled tapes.

For TAPELBL=BLP, the user must have the requested authority to the profile ICHBLP in the general resource class FACILITY. For TAPELBL=NL or BLP, the user will not be allowed to protect volumes with volume serial numbers in the format "Lnnnnn."

This parameter is primarily intended for use by data management routines to indicate the label type from the LABEL keyword on the JCL statement.

This parameter is valid only for CLASS='DATASET' and DSTYPE=T, or CLASS='TAPEVOL'. The default is TAPELBL=STD.

,STATUS=NONE
,STATUS=ERASE

Specifies whether or not RACHECK is to return the erase status of the given data set. This parameter is valid only for CLASS='DATASET' and a DSTYPE value other than T. The default is STATUS=NONE.

Parameters For RELEASE = 1.6 and Later

The RELEASE values for which a specific parameter is valid are marked with an 'X'.

Parameter	RELEASE = 1.6	RELEASE = 1.7
ACEE =	X	X
ACCLVL =	X	X
APPL =	X	X
ATTR =	X	X
CLASS =	X	X
DSTYPE = N, V, or M	X	X
DSTYPE = T		X
ENTITY =	X	X
FILESEQ =		X
GENERIC =	X	X
INSTLN =	X	X
LOG =	X	X
OLDVOL =	X	X
OWNER =	X	X
PROFILE =	X	X
RACFIND =	X	X
RELEASE =	X	X
STATUS =		X
TAPELBL =		X
VOLSER =	X	X

Figure 9. RACHECK Parameters for RELEASE = 1.6 and Later

Return Codes and Reason Codes

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

Hexadecimal Code	Meaning
00	The user is authorized by RACF to obtain use of a RACF-protected resource. Register 0 contains one of the following reason codes: 00 indicates a normal completion. 04 indicates STATUS=ERASE was specified and the data set is to be erased when scratched. Or the warning status of the resource was requested by the RACHECK issuer setting bit '10' at offset 12 decimal in the RACHECK parameter list and the resource is in warning mode.
04	The specified resource is not protected by RACF. Register 0 contains the following reason code: 00 indicates a normal completion.
08	The user is **not** authorized by RACF to obtain use of the specified RACF-protected resource. Register 0 contains the following reason code: 00 indicates a normal completion. 04 indicates STATUS=ERASE was specified and the data set is to be erased when scratched. 08 indicates DSTYPE=T or CLASS='TAPEVOL' was specified and the user is not authorized to use the specified volume. 0C indicates the user is not authorized to use the data set. 10 indicates DSTYPE=T or CLASS='TAPEVOL' was specified and the user is not authorized to specify LABEL=(,BLP).
0C	The OLDVOL specified was not part of the multivolume data set defined by VOLSER, or it was not part of the same tape volume defined by ENTITY.
64	Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACHECK macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

Example

Operation: Perform RACF authorization checking using the standard form, for a non-VSAM data set residing on the volume pointed to by register 8. Register 7 points to the data set name and the RACF user is requesting the highest level of control over the data set. The "RACF-indicated" bit in the data set's DSCB is on. Logging and statistics updates are **not** to be done.

```
RACHECK  ENTITY=((R7)),VOLSER=(R8),CLASS='DATASET',                X
         ATTR=ALTER,RACFIND=YES,LOG=NOSTAT
```

RACHECK (List Form)

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

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

PROFILE=*profile addr* ENTITY=(*resource name addr*) ENTITY=(*resource name addr*,CSA)	*profile addr:* A-type address. *resource name addr:* A-type address. **Note:** PROFILE or ENTITY is required on either the list or the execute form of the macro.
,VOLSER=*vol addr*	*vol addr:* A-type address. **Note:** VOLSER is required on either the list or the execute form of the macro, but only for CLASS='DATASET' and DSTYPE not equal to M when a discrete profile name is used. If required, VOLSER must be specified on either the list or the execute form of the macro.
,CLASS='*classname*' ,CLASS=*class name addr*	*'classname':* 1-8 character name. *class name addr:* A-type address. **Default:** CLASS='DATASET'
,RELEASE=*number*	*number:* 1.6 or 1.7 **Default:** RELEASE=1.6
,ATTR=READ ,ATTR=UPDATE ,ATTR=CONTROL ,ATTR=ALTER ,ATTR=*reg*	*reg:* register (2) - (12). **Default:** ATTR=READ
,DSTYPE=N ,DSTYPE=V ,DSTYPE=M ,DSTYPE=T	**Default:** DSTYPE=N
,INSTLN=*parm list addr*	*parm list addr:* A-type address.
,LOG=ASIS ,LOG=NOFAIL ,LOG=NONE ,LOG=NOSTAT	**Default:** LOG=ASIS
,OLDVOL=*old vol addr*	*old vol addr:* A-type address.
,APPL='*applname*' ,APPL=*applname addr*	*applname addr:* A-type address.
,ACEE=*acee addr*	*acee addr:* A-type address.

,OWNER=*owner ID addr*	*owner ID addr:* A-type address.
,ACCLVL=(*access value addr*) ,ACCLVL=(*access value addr*, *parm list addr*)	*access value addr:* A-type address or register (2)-(12). *parm list addr:* A-type address or register (2)-(12).
,RACFIND=YES ,RACFIND=NO	
,GENERIC=YES ,GENERIC=ASIS	**Default:** GENERIC=ASIS
,FILESEQ=*reg* ,FILESEQ=*number*	*reg:* register (2) - (12). *number:* 1-9999
,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default:** TAPELBL=STD
,STATUS=NONE ,STATUS=ERASE	**Default:** STATUS=NONE
,MF=L	

The parameters are explained under the standard form of the RACHECK macro instruction with the following exception:

,MF=L
specifies the list form of the RACHECK macro instruction.

RACHECK (Execute Form)

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

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

PROFILE=*profile addr* ENTITY=(*resource name addr*) ENTITY=(*resource name addr*,CSA)	*profile addr:* RX-type address, or register (2) - (12). *resource name addr:* RX-type address, or register (2) - (12). **Note:** PROFILE or ENTITY is required on either the list or the execute form of the macro.
,VOLSER=*vol addr*	*vol addr:* RX-type address, or register (2) - (12). **Note:** VOLSER is required on either the list or the execute form of the macro, but only for CLASS='DATASET' and DSTYPE not equal to M when a discrete profile name is used. If required, VOLSER must be specified on either the list or the execute form of the macro.
,CLASS='*classname*' ,CLASS=*class name addr*	*'classname':* 1-8 character name. *class name addr:* RX-type address, or register (2) - (12). **Default:** CLASS='DATASET'
,RELEASE=(*number*,CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number:* 1.6 or 1.7 **Default:** RELEASE=1.6
,ATTR=READ ,ATTR=UPDATE ,ATTR=CONTROL ,ATTR=ALTER ,ATTR=*reg*	*reg:* register (2) - (12). **Default:** ATTR=READ
,DSTYPE=N ,DSTYPE=V ,DSTYPE=M ,DSTYPE=T	**Default:** DSTYPE=N
,INSTLN=*parm list addr*	*parm list addr:* RX-type address, or register (2) - (12).
,LOG=ASIS ,LOG=NOFAIL ,LOG=NONE ,LOG=NOSTAT	**Default:** LOG=ASIS
,OLDVOL=*old vol addr*	*old vol addr:* RX-type address, or register (2) - (12).
,APPL='*applname*' ,APPL=*applname addr*	*applname addr:* RX-type address, or register (2) - (12).
,ACEE=*acee addr*	*acee addr:* RX-type address, or register (2) - (12).
,OWNER=*owner ID addr*	*owner ID addr:* RX-type address, or register (2) - (12).

,ACCLVL=(*access value addr*) ,ACCLVL=(*access value addr, parm list addr*)	*access value addr:* RX-type address or register (2)-(12). RX-type address or register (2)-(12). *parm list addr:*
,RACFIND=YES ,RACFIND=NO	
,GENERIC=YES ,GENERIC=ASIS	**Default:** GENERIC=ASIS
,FILESEQ=*reg* ,FILESEQ=*number*	*reg:* register (2) - (12). *number:* 1-9999
,TAPELBL=STD ,TAPELBL=BLP ,TAPELBL=NL	**Default:** TAPELBL=STD
,STATUS=NONE ,STATUS=ERASE	**Default:** STATUS=NONE
,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 RACHECK macro instruction with the following exceptions:

,MF=(E,*ctrl addr*)
specifies the execute form of the RACHECK macro instruction.

,RELEASE=(*number*,CHECK)
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

Certain parameters can be specified only with particular releases. For instance, to use the RACHECK release 1.7 parameter FILESEQ you must be using RACF 1.7 on your system and specify RELEASE=1.7. If you specify a parameter with an incompatible release level, the parameter will not be accepted by the macro processing. An error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 9 on page 2-292.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACHECK macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done. Instead, a return code X'64' will be generated.

RACINIT - Identify a RACF-Defined User

The RACINIT macro instruction is used to provide Resource Access Control Facility (RACF) user identification and verification. The macro instruction identifies a user and verifies that the user is defined to RACF and has supplied a valid password and/or operator identification card (OIDCARD parameter).

To issue the RACINIT macro instruction with the NEWPASS keyword, the calling module must be authorized (APF-authorized, in system key 0-7, or in supervisor state). To issue the RACINIT macro without the NEWPASS keyword, the calling module must either be authorized (APF-authorized, in system key 0-7, or in supervisor state) or in the RACF-authorized caller table and fetched from an authorized library.

Note: Only callers in 24-bit addressing mode can issue this macro. Callers executing in 31-bit addressing mode, who want to use the RACINIT function, can code the RACROUTE macro.

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

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

USERID=*userid addr* ,PASSWRD=*password addr*	*userid addr:* A-type address, or register (2) - (12). *password addr:* A-type address, or register (2) - (12).
,START=*procname addr*	*procname addr:* A-type address, or register (2) - (12).
,NEWPASS=*new password addr*	*new password addr:* A-type address, or register (2) - (12).
,GROUP=*group addr*	*group addr:* A-type address, or register (2) - (12). **Default:** zero.
,PGMNAME=*programmer name addr*	*programmer name addr:* A-type address, or register (2) - (12).
,ACTINFO=*account addr*	*account addr:* A-type address, or register (2) - (12).
,OIDCARD=*oid addr*	*oid addr:* A-type address, or register (2) - (12).
,TERMID=*terminal addr*	*terminal addr:* A-type address, or register (2) - (12).
,JOBNAME=*jobname addr*	*jobname addr:* A-type address, or register (2) - (12).
,ENVIR=CREATE ,ENVIR=CHANGE ,ENVIR=DELETE	**Default:** ENVIR=CREATE **Notes:** 1. ENVIR=CHANGE may not be specified with USERID=, PASSWRD=, START=, NEWPASS=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters. 2. ENVIR=DELETE may not be specified with APPL=, USERID=, PASSWRD=, START=, NEWPASS=, GROUP=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters.
,INSTLN=*parm list addr*	*parm list addr:* A-type address, or register (2) - (12).
,APPL='*applname*' ,APPL=*applname addr*	*applname addr:* A-type address, or register (2) - (12).
,ACEE=*acee addr*	*acee addr:* A-type address, or register (2) - (12).
,SUBPOOL=*subpool number*	*subpool number:* decimal digit 0-255.
,SMC=YES ,SMC=NO	**Default:** SMC=YES
,PASSCHK=YES ,PASSCHK=NO	**Default:** PASSCHK=YES
,ENCRYPT=YES ,ENCRYPT=NO	**Default:** ENCRYPT=YES
,RELEASE=*number*	*number:* 1.6 or 1.7 **Default:** RELEASE=1.6
,STAT=ASIS ,STAT=NO	**Default:** STAT=ASIS
,LOG=ASIS ,LOG=ALL	**Default:** LOG=ASIS

The parameters are explained as follows:

USERID = *userid addr*
specifies the user identification of the user who has entered the system. The address points to a one-byte length field, followed by the userid.

,PASSWRD = *password addr*
specifies the currently defined password of the user who has entered the system. The address points to a one-byte length field, followed by the password.

,START = *procname addr*
specifies the PROC name of a started task. The address points to an eight-byte area containing the PROC name (left-justified and padded with blanks, if necessary). The START parameter is not used by RACINIT authorization checking, but it is passed to the installation exit routine.

,NEWPASS = *new password addr*
specifies the password which is to replace the user's currently defined password. The address points to a one-byte length field, followed by the password.

,GROUP = *group addr*
specifies the group specified by the user who has entered the system. The address points to a one-byte length field, followed by the group name.

,PGMNAME = *programmer name addr*
specifies the address of the name of the user who has entered the system. This twenty byte area is passed to the RACINIT installation exit routine; it is not used by the RACINIT routine.

,ACTINFO = *account addr*
specifies the address of a field containing accounting information. This 144 byte area is passed to the RACINIT installation exit routine; it is not used by the RACINIT routine. The accounting field, if supplied, should have the following format:

- First byte of field contains the number (binary) of accounting fields.
- Following bytes contain accounting fields, where each entry for an accounting field contains a one-byte length field, followed by the field.

,OIDCARD = *oid addr*
specifies the address of the currently defined operator identification card of the user who has entered the system. The address points to a one-byte length field, followed by the operator ID card.

,TERMID = *terminal addr*
specifies the address of the identifier for the terminal through which the user is accessing the system. The address points to an eight byte area containing the terminal identifier. The area must reside in a non-task-related storage subpool.

,JOBNAME = *jobname addr*
specifies the address of the JOB name of a background job. The address points to an eight byte area containing the JOB name (left justified and padded with blanks, if necessary). The JOBNAME parameter is not used by RACINIT authorization checking, but it is passed to the installation exit routine.

,ENVIR=CREATE
,ENVIR=CHANGE
,ENVIR=DELETE

specifies the action to be performed by the user initialization component regarding the accessor environment element (ACEE):

CREATE -	The user should be verified and an ACEE created.
CHANGE -	The ACEE should be modified according to other parameters specified on RACINIT.
DELETE -	The ACEE should be deleted. This parameter should only be used if a previous RACINIT has completed successfully.

,INSTLN=*parm list addr*

specifies the address of an area containing parameter information meaningful to the RACINIT installation exit routine. This area is passed to the installation exit when the exit routine is given control from the RACINIT routine.

The INSTLN parameter can be used by an installation having a user verification or job initiation application, and wanting to pass information from one installation module to the RACINIT installation exit routine.

,APPL=*'applname'*
,APPL=*applname addr*

specifies the name of the application issuing the RACINIT. If an address is specified, the address must point to an 8-byte application name, left justified and padded with blanks, if necessary.

,ACEE=*acee addr*

specifies the address of the ACEE.

For ENVIR=DELETE: specifies the address of a fullword that contains the address of the accessor environment element (ACEE) to be deleted. If ACEE= is not specified, and the TCBSENV field for the task using the RACINIT is non-zero, the ACEE pointed to by the TCBSENV is deleted, and TCBSENV is set to zero. If the TCBSENV and ASXBSENV fields both point to the same ACEE, then ASXBSENV is also set to zero. If no ACEE address is passed, and TCBSENV is zero, the ACEE pointed to by ASXBSENV is deleted, and ASXBSENV is set to zero.

For ENVIR=CHANGE: specifies the address of a fullword that contains the address of the accessor environment element (ACEE) to be changed. If ACEE= is not specified, and the TCBSENV field for the task using the RACINIT is non-zero, the ACEE pointed to by the TCBSENV is changed. If TCBSENV is 0, then the ACEE pointed to by ASXBSENV is changed.

For ENVIR=CREATE: specifies the address of a full word into which the RACINIT function will place the address of the ACEE created. If an ACEE is not specified, the address of the newly created ACEE is stored in the TCBSENV field of the task control block. If the ASXBSENV field is set to binary zeros, the new ACEE address is also stored in the ASXBSENV field of the ASXB. If the ASXBSENV field is non-zero, it is not modified. The TCBSENV field is set unconditionally.

Notes:

1. *If you omit USERID, GROUP, and PASSWRD and if you code ENVIR=CREATE or if ENVIR=CREATE is used as the default, you will receive a return code of X'00' and obtain an ACEE that contains an * (X'5C') in place of the USERID and group name.*

2. *If ACEE is specified with ENVIR=CREATE, RACF suppresses the creation of a modeling table (MDEL) to reduce the amount of CSA and/or LSQA storage required by IMS/VS and CICS/VS installations.*

,SUBPOOL=*subpool number*
specifies the storage subpool from which the ACEE and related storage are obtained.

,SMC=YES
,SMC=NO
specifies the use of the step-must-complete function of RACINIT processing. SMC=YES specifies that RACINIT processing should continue to place other tasks for the step non-dispatchable. SMC=NO specifies that the step-must-complete function is not used.

Note: SMC=NO should not be used if DADSM ALLOCATE/SCRATCH functions execute simultaneously in the same address space as the RACINIT function.

,PASSCHK=YES
,PASSCHK=NO
specifies whether or not the user's password is to be verified. PASSCHK=YES specifies that RACINIT verifies the user's password. PASSCHK=NO specifies that the user's password is not verified.

,ENCRYPT=YES
,ENCRYPT=NO
specifies whether or not RACINIT will encrypt the old password, the new password, and the OIDCARD data passed to it.

YES signifies that the data specified by the PASSWRD, NEWPASS, and OIDCARD keywords are not pre-encrypted. RACINIT encrypts the data before storing it in the user profile or using it to compare against stored data. ENCRYPT=YES is the default for this keyword.

NO signifies that the data specified by the PASSWRD, NEWPASS, and OIDCARD keywords are already encrypted. RACINIT bypasses the encryption of this data before storing it in, or comparing it against, the user profile.

Note: The exit routine ICHDEX01 can also perform the encryption.

,RELEASE=*number*
specifies the RACF release level of the parameter list to be generated by this macro.

Certain parameters can be specified only with particular releases. For instance, to use the RACINIT release 1.7 parameter STAT you must be using RACF 1.7 on your system and specify RELEASE=1.7. If you specify a parameter with an incompatible release level, the parameter will not be accepted by the macro processing. An error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 10 on page 2-304.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACINIT macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

,STAT=ASIS
,STAT=NO
specifies whether the statistics controlled by the installation's options on the RACF SETROPTS command are to be maintained or ignored for this execution of RACINIT. This parameter also controls whether a message is to be issued when the logon is successful.

Note: Messages are always issued if the RACINIT processing is unsuccessful.

If STAT=ASIS is specified or taken by default, the messages and statistics are controlled by the installation's current options on the RACF SETROPTS command.

If STAT=NO is specified, the statistics are not updated. And, if the logon is successful, no message is issued.

The default is STAT=ASIS.

,LOG=ASIS
,LOG=ALL
specifies when log records are to be generated.

If LOG=ASIS is specified or defaulted to, only those attempts to create an ACEE that fail will generate RACF log records.

If LOG=ALL is specified, any request to create an ACEE, regardless of whether it succeeds or fails, will generate a RACF log record. The default is LOG=ASIS.

Parameters For RELEASE=1.6 and Later

The RELEASE values for which a specific parameter is valid are marked with an 'X'.

Parameter	RELEASE= 1.6	RELEASE= 1.7
ACEE=	X	X
ACCTINFO=	X	X
APPL=	X	X
ENCRYPT=	X	X
ENVIR=	X	X
GROUP=	X	X
INSTLN=	X	X
JOBNAME=	X	X
LOG=		X
NEWPASS=	X	X
OIDCARD=	X	X
PASSCHK=	X	X
PASSWRD=	X	X
PGMNAME=	X	X
RELEASE=	X	X
SMC=	X	X
START=	X	X
STAT=		X
SUBPOOL=	X	X
TERMID=	X	X
USERID=	X	X

Figure 10. RACINIT Parameters for RELEASE=1.6 and Later

Return Codes and Reason Codes

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

Hexadecimal Code	Meaning
00	RACINIT has completed successfully.
04	The user profile is not defined to RACF.
08	The password is not authorized.
0C	The password has expired.
10	The new password is invalid.
14	The user is not defined to the group.
18	RACINIT was failed by the installation exit routine.
1C	The user access has been revoked.
20	RACF is not active.
24	The user's access to the specified group has been revoked.
28	OIDCARD parameter is required but not supplied.
2C	OIDCARD parameter is invalid for specified user.
30	The user is not authorized to use the terminal. Register 0 contains one of the following reason codes: 00 indicates a normal completion. 04 indicates the user is not authorized to access the system on this day, or at this time of day. 08 indicates the terminal may not be used on this day, or at this time of day.
34	The user is not authorized to use the application.
64	Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACINIT macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

RACINIT (List Form)

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

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

USERID=*userid addr* ,PASSWRD=*password addr*	*userid addr:* A-type address. *password addr:* A-type address.
,START=*procname addr*	*procname addr:* A-type address.
,NEWPASS=*new password addr*	*new password addr:* A-type address.
,GROUP=*group addr*	*group addr:* A-type address.
,PGMNAME=*programmer name addr*	*programmer name addr:* A-type address.
,ACTINFO=*account addr*	*account addr:* A-type address.
,OIDCARD=*oid addr*	*oid addr:* A-type address, or register (2) - (12).
,TERMID=*terminal addr*	*terminal addr:* A-type address.
,JOBNAME=*jobname addr*	*jobname addr:* A-type address.
,ENVIR=CREATE ,ENVIR=CHANGE ,ENVIR=DELETE	**Default:** ENVIR=CREATE **Notes:** 1. ENVIR=CHANGE may not be specified with USERID=, PASSWRD=, START=, NEWPASS=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters. 2. ENVIR=DELETE may not be specified with APPL=, USERID=, PASSWRD=, START=, NEWPASS=, GROUP=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters.
,INSTLN=*parm list addr*	*parm list addr:* A-type address.
,APPL='*applname*' ,APPL=*applname addr*	*applname addr:* A-type address.
,ACEE=*acee addr*	*acee addr:* A-type address.
,SUBPOOL=*subpool number*	*subpool number:* decimal digit 0-255.
,SMC=YES ,SMC=NO	**Default:** SMC=YES
,PASSCHK=YES ,PASSCHK=NO	**Default:** PASSCHK=YES
,ENCRYPT=YES ,ENCRYPT=NO	**Default:** ENCRYPT=YES
,RELEASE=*number*	*number:* 1.6 or 1.7 **Default:** RELEASE=1.6

,STAT = ASIS ,STAT = NO	**Default:** STAT = ASIS
,LOG = ASIS ,LOG = ALL	**Default:** LOG = ASIS
,MF = L	

The parameters are explained under the standard form of the RACINIT macro instruction, with the following exception:

,MF = L
specifies the list form of the RACINIT macro instruction.

RACINIT (Execute Form)

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

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

USERID=*userid addr* ,PASSWRD=*password addr*	*userid addr:* RX-type address, or register (2) - (12). *password addr:* RX-type address, or register (2) - (12).
,START=*procname addr*	*procname addr:* RX-type address, or register (2) - (12).
,NEWPASS=*new password addr*	*new password addr:* RX-type address, or register (2) - (12).
,GROUP=*group addr*	*group addr:* RX-type address, or register (2) - (12). **Default:** zero.
,PGMNAME=*programmer name addr*	*programmer name addr:* RX-type address, or register (2) - (12).
,ACTINFO=*account addr*	*account addr:* RX-type address, or register (2) - (12).
,OIDCARD=*oid addr*	*oid addr:* RX-type address, or register (2) - (12).
,TERMID=*terminal addr*	*terminal addr:* RX-type address, or register (2) - (12).
,JOBNAME=*jobname addr*	*jobname addr:* RX-type address, or register (2) - (12).
,ENVIR=CREATE ,ENVIR=CHANGE ,ENVIR=DELETE	**Default:** ENVIR=CREATE **Notes:** 1. ENVIR=CHANGE may not be specified with USERID=, PASSWRD=, START=, NEWPASS=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters. 2. ENVIR=DELETE may not be specified with APPL=, USERID=, PASSWRD=, START=, NEWPASS=, GROUP=, ACTINFO=, PGMNAME=, OIDCARD=, or TERMID= parameters.
,INSTLN=*parm list addr*	*parm list addr:* RX-type address, or register (2) - (12).
,APPL='*applname*' ,APPL=*applname addr*	*applname addr:* RX-type address, or register (2) - (12).
,ACEE=*acee addr*	*acee addr:* RX-type address, or register (2) - (12).
,SUBPOOL=*subpool number*	*subpool number:* decimal digit 0-255.
,SMC=YES ,SMC=NO	**Default:** SMC=YES
,PASSCHK=YES ,PASSCHK=NO	**Default:** PASSCHK=YES
,ENCRYPT=YES ,ENCRYPT=NO	**Default:** ENCRYPT=YES

,RELEASE=*(number*,CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number:* 1.6 or 1.7 **Default:** RELEASE=1.6
,STAT=ASIS ,STAT=NO	**Default:** STAT=ASIS
,LOG=ASIS ,LOG=ALL	**Default:** LOG=ASIS
,MF=(E,*ctrl addr)*	*cntl addr:* RX-type address, or register (1) or (2) - (12).

The parameters are explained under the standard form of the RACINIT macro instruction, with the following exception:

,MF=(E,*ctrl addr)*
specifies the execute form of the RACINIT macro instruction using a remote control program parameter list.

,RELEASE=(*number***,CHECK)**
,RELEASE=*number*
,RELEASE=(,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

Certain parameters can be specified only with particular releases. For instance, to use the RACINIT release 1.7 parameter STAT you must be using RACF 1.7 on your system and specify RELEASE=1.7. If you specify a parameter with an incompatible release level, the parameter will not be accepted by the macro processing. An error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 10 on page 2-304.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACINIT macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done. Instead, a return code X'64' will be generated.

RACLIST - Build In-Storage Profiles

RACLIST is used to build in-storage profiles for RACF defined resources. RACLIST processes only those resources described by class descriptors. The primary advantage of using the RACLIST macro is to use the resource grouping function and to improve resource authorization checking performance.

The module calling the RACLIST macro instruction must either be authorized (APF-authorized, in system key 0-7, or in supervisor state) or in the RACF-authorized caller table and fetched from an authorized library.

Notes:

1. *Only callers in 24-bit addressing mode can issue this macro. Callers executing in 31-bit addressing mode, who want to utilize the RACLIST function, can code the RACROUTE macro.*

2. *For RACF Version 1, Release 6, all parameters and parameter lists must reside below 16 megabytes.*

3. *For RACF Version 1, Release 7, all parameters and parameter lists are assumed to reside below 16 megabytes when the caller is executing in 24-bit addressing mode.*

The standard form of the RACLIST macro is written as follows:

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

CLASS = *'classname'* CLASS = *classname addr*	*classname addr:* A-type address or register (2) - (12).
,LIST = *list addr*	*list addr:* A-type address or register (2) - (12).
,ACEE = *acee addr*	*acee addr:* A-type address or register (2) - (12).
,INSTLN = *parm list addr*	*parm list addr:* A-type address or register (2) - (12).
,APPL *'applname'* ,APPL = *applname addr*	*applname addr:* A-type address or register (2) - (12).
,SUBPOOL = (*sub#1,sub#2*)	*sub#,sub#2:* decimal digit 0-255.
,ENVIR = CREATE ,ENVIR = DELETE	**Default:** ENVIR = CREATE
,OWNER = YES ,OWNER = NO	**Default:** OWNER = NO
,RELEASE = *number*	*number:* 1.6 or 1.7 **Default:** RELEASE = 1.6

The parameters are explained as follows:

CLASS = *'classname'*
CLASS = *classname addr*
specifies that RACLIST is to build an in-storage profile for the resources of the specified class. If an address is specified, the address must point to an 8-byte field containing the class name, left justified and padded with blanks, if necessary. The class name must be defined by a class descriptor; if not, the class is not considered to be defined.

,LIST = *addr*
specifies the address of a list of resource names for which RACLIST is to build the in-storage profiles. The list consists of a 2-byte field containing the number of the names in the list, followed by one or more variable length names. Each name consists of a 1-byte length field, followed by the name. A zero in the 2-byte field causes the operand to be omitted. If LIST= is omitted, in-storage profiles are built for all the profiles defined to RACF in the given class as well as each member for a resource grouping associated with the specified class.

Note: This operand can be specified only with ENVIR=CREATE. If ENVIR=DELETE is specified, the RACLIST macro instruction issues a return code of 18.

,ACEE = *acee addr*

specifies the address of the accessor control environment element (ACEE). The ACEE points to the in-storage profiles. If an ACEE is not specified, RACF uses the TASK ACEE pointer in the extended TCB called the TCBSENV. Otherwise, or if the TASK ACEE pointer is zero, RACF uses the main ACEE to obtain the list of the in-storage profiles. The main ACEE is pointed to by the ASXBSENV field of the address space extension block. If an ACEE is not specified and there is no main ACEE, the in-storage profiles are not constructed.

,INSTLN = *parm list addr*

specifies the address of an area that contains parameter information for the RACLIST installation exit. The address is passed to the installation exit when the exit is given control by the RACLIST routine. The INSTLN parameter can be used by an application or an installation program to pass information to the RACLIST installation exit.

,APPL = *'applname'*
,APPL = *applname addr*

specifies the name of the application requesting the authorization checking. This information is not used for the authorization checking process but is made available to the installation exit(s). If an address is specified, it should point to an 8-byte area containing the application name, left justified and padded with blanks, if necessary.

,SUBPOOL = *(sub#1,sub#2)*

specifies the subpool numbers of the storage into which the components of the in-storage profiles are to be built. Sub#1 represents the subpool of the profile index. Sub#2 represents the subpool of the profile proper. If the subpools are not specified they default to subpool 255. Registers can be used to specify sub#1 and sub#2.

,ENVIR = CREATE
,ENVIR = DELETE

specifies the action to be performed by the RACLIST macro.

CREATE - In-storage profiles for the specified class are to be built. The RACLIST function issues a return code of 18, if an in-storage list currently exists for the specified class.

DELETE - The in-storage profiles for the specified class are to be freed. If class is not specified, the in-storage profiles for all classes are freed.

Note: It is the responsibility of the user issuing the RACLIST macro to assure that no multi-tasking that results in the issuing of a RACHECK, FRACHECK, RACINIT, or RACLIST macro instruction occurs at the same time that the RACLIST occurs.

,OWNER=YES
,OWNER=NO

specifies that the resource owner is to be placed in the profile access list with the ALTER authority. If the OWNER= operand is omitted, the default is NO.

,RELEASE=*number*

specifies the RACF release level of the parameter list to be generated by this macro.

Certain parameters can be specified only with particular releases. If you specify a parameter with an incompatible release level, the parameter will not be accepted by the macro processing. An error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 11 on page 2-314.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACLIST macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

Parameters For RELEASE = 1.6 and Later

The RELEASE values for which a specific parameter is valid are marked with an 'X'.

Parameter	RELEASE = 1.6	RELEASE = 1.7
ACEE =	X	X
APPL =	X	X
CLASS =	X	X
ENVIR =	X	X
INSTLN =	X	X
LIST =	X	X
OWNER =	X	X
RELEASE =	X	X
SUBPOOL =	X	X

Figure 11. RACLIST Parameters for RELEASE = 1.6 and Later

Return Codes and Reason Codes

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

Hexadecimal Code	Meaning
00	RACLIST function completed successfully.
04	Unable to perform the requested function. Register 0 contains additional codes as follows: 0 - Unable to establish an ESTAE environment. 1 - The function code (the third byte of the parameter list) does not represent a valid function. '01' represents the RACF manager; '02' represents the RACLIST macro.
08	The specified class is not defined to RACF.
0C	An error was encountered during RACLIST processing.
10	RACF and/or the resource class is not active.
14	RACLIST installation exit error occurred.
18	Parameter list error.
1C	RACF CVT does not exist (RACF is not installed) or an insufficient level of RACF is installed.
64	Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACLIST macro; however, the list form of the macro does not have the proper RELEASE parameter. Macro processing terminates.

Note: If the resource class specified by the CLASS = operand is inactive, RACLIST does not build the in-storage profiles and a code of 0C is returned. If the resource group class is not active, RACLIST builds an in-storage profile but only from the individual resource profiles; resource group profiles are ignored.

RACLIST (List Form)

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

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

CLASS = *'classname'* CLASS = *classname addr*	*classname addr:* A-type address.
,LIST = *list addr*	*list addr:* A-type address.
,ACEE = *acee addr*	*acee addr:* A-type address.
,INSTLN = *parm list addr*	*parm list addr:* A-type address.
,APPL = *'applname'* ,APPL = *applname addr*	*applname addr:* A-type address.
,SUBPOOL = (*sub#1,sub#2*)	*sub#1,sub#2:* decimal digit 0-255. **Default:** 255.
,ENVIR = CREATE ,ENVIR = DELETE	**Default:** ENVIR = CREATE
,OWNER = YES ,OWNER = NO	**Default:** OWNER = NO
,RELEASE = *number*	*number:* 1.6 or 1.7 **Default:** RELEASE = 1.6
,MF = L	

The parameters are explained under the standard form of the RACLIST macro instruction with the following exception:

,MF = L
specifies the list form of the RACLIST macro instruction.

RACLIST (Execute Form)

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACLIST.
RACLIST	
b	One or more blanks must follow RACLIST.
CLASS = *classname addr*	*classname addr:* RX-type address or register (2) - (12).
,LIST = *list addr*	*list addr:* RX-type address or register (2) - (12).
,ACEE = *acee addr*	*acee addr:* RX-type address or register (2) - (12).
,INSTLN = *parm list addr*	*parm list addr:* A-type address or register (2) - (12).
,APPL = *applname addr*	*applname addr:* RX-type address or register (2) - (12).
,SUBPOOL = (*sub#1,sub#2*)	*sub#1,sub#2:* decimal digit 0-255.
,ENVIR = CREATE ,ENVIR = DELETE	
,OWNER = YES ,OWNER = NO	
,RELEASE = (*number*,CHECK) ,RELEASE = *number* ,RELEASE = (,CHECK)	*number:* 1.6 or 1.7 **Default:** RELEASE = 1.6
,MF = (E,,*ctrl addr*)	*ctrl addr:* RX-type address or register (2) - (12).

The parameters are explained under the standard form of the RACLIST macro instruction with the following exception:

,MF = (E,*ctrl addr)*
specifies the execute form of the RACLIST macro instruction using a remote control program parameter list.

,RELEASE=(*number*,CHECK)
,RELEASE=*number*
,RELEASE=(,CHECK)

specifies the RACF release level of the parameter list to be generated by this macro.

Certain parameters can be specified only with particular releases. If you specify a parameter with an incompatible release level, the parameter will not be accepted by the macro processing. An error message will be issued at assembly time. For the parameters that are valid for RELEASE=1.6 and later, see Figure 11 on page 2-314.

The default is RELEASE=1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACLIST macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done. Instead, a return code of X'64' will be generated.

RACROUTE - MVS Router Interface

The RACROUTE macro instruction is used to invoke the System Authorization Facility (SAF) MVS router, which conditionally directs control to the Resource Access Control Facility (RACF) when RACF is present.

You can use RACROUTE to access the functions that are provided by the following RACF macros: RACDEF, RACINIT, RACXTRT, RACLIST, RACHECK, and FRACHECK. In coding the RACROUTE macro instruction to access a particular RACF macro function, you must also use the necessary parameters from that macro on the RACROUTE macro instruction. For example, if you code RACROUTE to access the RACHECK function, you must code REQUEST=AUTH and any other required parameters and any optional ones you need from the RACHECK macro. RACROUTE validates that only the parameters applicable to the RACHECK macro have been coded.

This macro is also described in *Supervisor Services and Macro Instructions* with the exception of the REQUEST=DEFINE, REQUEST=VERIFY, REQUEST=LIST, and REQUEST=EXTRACT parameters, which are restricted in use to programs that are authorized (AFP authorized, in system key 0-7, or in supervisor state).

Notes:

1. *For RACF Version 1 Release 6, all parameters and parameter lists must reside below 16 megabytes.*

2. *For RACF Version 1 Release 7:*
   If a caller is executing in 24-bit addressing mode, all parameters and parameter lists are assumed to reside below 16 megabytes. If a caller, however, is executing in 31-bit addressing mode, and is calling RACF via the RACROUTE macro instruction, RACF will assume that all parameters and parameter lists may reside above the 16 megabytes (that is, that all parameter addresses are true 31-bit addresses).

   All parameter lists generated by the RACROUTE macro instruction are in a format that allows compiled code to be moved above 16 megabytes without recompilation.

   This 31-bit support is available only when RACF is called via the RACROUTE, FRACHECK, or RACSTAT macro instructions. Any caller that uses the RACINIT, RACDEF, RACLIST or RACHECK macro instructions may be in 24-bit addressing mode only. RACF does not support those callers in 31-bit mode. RACHECK macro instruction may be in 24-bit addressing mode only. RACF does not support this caller in 31-bit mode.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST=AUTH REQUEST=FASTAUTH REQUEST=DEFINE REQUEST=VERIFY REQUEST=LIST REQUEST=EXTRACT	
,REQSTOR=*reqstor addr*	*reqstor addr:* A-type address or register (2) - (12). **Default:** zero. **Note:** If REQSTOR= is coded and RACF is installed, the RACF router table must be updated to match the operand.
,SUBSYS=*subsys addr*	*subsys addr:* A-type address or register (2) - (12). **Note:** If SUBSYS= is coded and RACF is installed, the RACF router table must be updated to match the operand.
,WORKA=*work area addr*	*work area addr:* A-type address or register (2) - (12).
,RELATED=*value*	*value:* Any valid macro keyword specified.
,ENVIR=VERIFY	**Note:** ENVIR can be coded only if REQUEST=VERIFY is coded.
,LOC=BELOW ,LOC=ANY ,LOC=ABOVE	**Default:** See parameter description. **Note:** LOC can be coded only if REQUEST=VERIFY or REQUEST=LIST is coded.
In addition to the parameters described above, all parameters valid on the RACDEF, RACLIST, RACINIT, RACXTRT, RACHECK, and FRACHECK macros are permitted on the RACROUTE macro. Depending on the parameter REQUEST=, some of these are required, some optional, and some are invalid.	

The parameters are explained as follows:

REQUEST=AUTH
REQUEST=FASTAUTH
REQUEST=DEFINE
REQUEST=VERIFY
REQUEST=LIST
REQUEST=EXTRACT

specifies a code that determines the RACF parameter list to be issued internally as well as the RACF routine to receive control. The permissible codes and their associated RACF macros are as follows:

AUTH -- RACHECK
FASTAUTH -- FRACHECK
DEFINE -- RACDEF
VERIFY -- RACINIT
LIST -- RACLIST
EXTRACT -- RACXTRT

For RACROUTE to work correctly, once you have chosen a REQUEST code you must also code (on the RACROUTE macro) the parameters that belong to the associated macro instruction. Please see the associated macro for the necessary parameters.

Notes:

1. *Data areas returned by RACF to the caller will be either above or below the 16-megabyte line, depending upon the caller's addressing mode and the data area in question.*

2. *Unless the caller specifies the ACEE= parameter on a "RACROUTE REQUEST=VERIFY,ENVIR=CREATE" macro instruction, the ACEE will always be placed below the 16-megabyte line.*

3. *If the caller specifies the ACEE= parameter, and is executing in 31-bit addressing mode and does not specify LOC=BELOW on the RACROUTE macro instruction, the ACEE will be placed, if possible, above the 16-megabyte line.*

4. *If the ACEE is below the 16-megabyte line, any area chained off an ACEE (for example, RACLIST profiles, list-of-groups table) will be placed below the 16-megabyte line. Otherwise, the area will be placed above the line. However, a caller executing in 31-bit mode may issue a REQUEST=LIST with LOC=ABOVE, and the profiles will be placed above the line, if possible, even if the ACEE is below the line.*

5. *If the caller requests that RACF return an in-storage profile in CSA as part of a "RACROUTE REQUEST=AUTH," the profile will be returned in storage below the 16-megabyte line if the related ACEE is located below the line. Otherwise, the area will be located above the line.*

6. *The area returned by a "RACROUTE REQUEST=EXTRACT" request will be located below the 16-megabyte line if the related ACEE is located below the line. Otherwise, the area will be located above the line.*

,REQSTOR=*reqstor addr*
specifies the address of an 8-byte character field containing the control point name (this address identifies a unique control point within a set of control points that exists in a subsystem). If this operand is coded and RACF is installed, the RACF router table must be changed to match the operand. If the table is not updated, the default to bypass RACF processing is taken. For a description of the RACF router table and the macro used to update it, see *SPL: Resource Access Control Facility (RACF)*.

If this operand is omitted, a string of eight blanks is assumed.

,SUBSYS=*subsys addr*
specifies the address of an 8-byte character field containing the calling subsystem's name, version, and release level. If this operand is coded and RACF is installed, the RACF router table must be changed to match the operand. If the table is not updated, the default to bypass RACF processing is taken. For a description of the RACF router table and the macro used to update it, see *SPL: Resource Access Control Facility (RACF)*.

If this operand is omitted, a string of eight blanks is assumed.

,WORKA=*work area addr*
specifies the address of a 512-byte work area for use by the MVS router and the RACF front end routine.

,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 is at the discretion of the user, and can be any valid coding value.

,ENVIR=VERIFY

specifies that only a user verification is to be made optionally combined with updating the user's password. The installation may handle this request through a System Authorization Facility (SAF) installation exit. If this is not done, the RACROUTE caller receives a return code of 4 with RACF return and reason codes of zero (the request is not processed by the RACF SVC).

,LOC=BELOW
,LOC=ANY
,LOC=ABOVE

LOC can be coded only if REQUEST=VERIFY or REQUEST=LIST is coded.

For REQUEST=VERIFY:
specifies whether the ACEE and related data areas are to be allocated storage below 16 megabytes (LOC=BELOW), or anywhere (LOC=ANY).

If any of the following is true, LOC=BELOW is the default, and LOC=ANY is ignored if specified:

- The ACEE= parameter is not coded.
- The caller is executing in 24-bit mode.

In all other cases, the default is LOC=ANY.

Note: LOC=ABOVE is invalid for REQUEST=VERIFY.

For REQUEST=LIST:
specifies whether the RACLIST profiles are to reside where the ACEE is located, above or below 16 megabytes (LOC=ANY), or whether the RACLIST profiles are to reside above 16 megabytes (LOC=ABOVE), if possible, even if the ACEE is below 16 megabytes.

Notes:

1. *LOC=BELOW is invalid for REQUEST=LIST.*

2. *LOC=ANY does not guarantee that storage is allocated above 16 megabytes. If any installation SAF or RACF exit routines execute in 24-bit mode, the storage will be below 16 megabytes.*

Return Codes and Reason Codes

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

Hexadecimal Code	Meaning
00	The requested security function has completed successfully. In addition, if the requested function was 'AUTH', the authorization request was accepted.
04	The requested function has not been processed. In addition, if the request was 'AUTH', the MVS router could neither accept nor fail the request. The following are possible reasons for a request not being processed. - The MVS router is not active. - The RACF front end routine detected that a null action was requested for the specified request type, resource type, and subsystem ID. - The request/resource/subsystem combination could not be found in the router table. - RACF is not active on the system, and RACFIND=YES was not specified, and there is no RACROUTE installation exit routine (or an exit originated a return code of 4). - RACF is active on the system, but no profile exists for the specified resource.
08	The requested function was processed by RACF, the MVS router, or the router exit (ICHRTX00) and failed. If the requested function was 'AUTH', the authorization request has been failed. If RACF is inactive for an 'AUTH' request with RACFIND=YES, then the MVS router fails the request. The RACF or router exit return code and reason codes are returned in the first two words of the RACROUTE input parameter list.

Example 1

Operation: Invoke the MVS router to perform authorization checking using the standard form, for a non-VSAM data set residing on the volume pointed to by register 8. Register 7 points to the data set name and the RACF user is requesting the highest level of control over the data set. The "RACF-indicated" bit in the data set's DSCB is on.

```
RACROUTE  REQUEST=AUTH,WORKA=RACWK,ENTITY=((R7)),                X
          VOLSER=(R8),CLASS='DATASET',ATTR=ALTER,                X
          RACFIND=YES
  .
  .
RACWK     DS   CL512
```

Example 2

Operation: Invoke the MVS router to perform authorization checking using the standard form, for an IMS/VS transaction pointed to by register 5. The user requests only read access. The request is issued on behalf of the IMS/VS subsystem.

```
RACROUTE  REQUEST=FASTAUTH,SUBSYS=SUBIMS,                        X
          WORKA=RACWK,ENTITY=(R5),                               X
          CLASS='TIMS',WKAREA=FRACWK,                            X
          ATTR=READ
  .
  .
  .
SUBIMS    DC   CL8'IMS'
FRACWK    DS   16F
RACWK     DS   CL512
```

RACROUTE (List Form)

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST=AUTH REQUEST=FASTAUTH REQUEST=DEFINE REQUEST=VERIFY REQUEST=LIST REQUEST=EXTRACT	
,REQSTOR=*reqstor addr*	*reqstor addr:* A-type address. **Default:** zero. **Note:** If REQSTOR= is coded and RACF is installed, the RACF router table must be updated to match the operand.
,SUBSYS=*subsys addr*	*subsys addr:* A-type address. **Note:** If SUBSYS= is coded and RACF is installed, the RACF router table must be updated to match the operand.
,WORKA=*work area addr*	*work area addr:* A-type address or register (2) - (12).
,RELATED=*value*	*value:* Any valid macro keyword specified.
,ENVIR=VERIFY	**Note:** ENVIR can be coded only if REQUEST=VERIFY is coded.
,LOC=BELOW ,LOC=ANY ,LOC=ABOVE	**Default:** See parameter description. **Note:** LOC can be coded only if REQUEST=VERIFY or REQUEST=LIST is coded.
,MF=L	

In addition to the parameters described above, all parameters valid on the RACDEF, RACLIST, RACINIT, RACXTRT, RACHECK, and FRACHECK macros are permitted on the RACROUTE macro. Depending on the parameter REQUEST=, some of these are required, some optional, and some are invalid.

The parameters are explained under the standard form of the RACROUTE macro instruction with the following exception:

,MF=L
specifies the list form of the RACROUTE macro instruction.

RACROUTE (Execute Form)

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RACROUTE.
RACROUTE	
b	One or more blanks must follow RACROUTE.
REQUEST=AUTH REQUEST=FASTAUTH REQUEST=DEFINE REQUEST=VERIFY REQUEST=LIST REQUEST=EXTRACT	
,REQSTOR=*reqstor addr*	*reqstor addr:* RX-type address or register (2) - (12). **Default:** zero. **Note:** If REQSTOR= is coded and RACF is installed, the RACF router table must be updated to match the operand.
,SUBSYS=*subsys addr*	*subsys addr:* RX-type address or register (2) - (12). **Note:** If SUBSYS= is coded and RACF is installed, the RACF router table must be updated to match the operand.
,WORKA=*work area addr*	*work area addr:* RX-type address or register (2) - (12).
,RELATED=*value*	*value:* Any valid macro keyword specified.
,ENVIR=VERIFY	**Note:** ENVIR can be coded only if REQUEST=VERIFY is coded.
,LOC=BELOW ,LOC=ANY ,LOC=ABOVE	**Default:** See parameter description. **Note:** LOC can be coded only if REQUEST=VERIFY or REQUEST=LIST is coded.
,MF=(E,*ctrl addr*)	*ctrl addr:* RX-type address or register (1).
In addition to the parameters described above, all parameters valid on the RACDEF, RACLIST, RACINIT, RACXTRT, RACHECK, and FRACHECK macros are permitted on the RACROUTE macro. Depending on the parameter REQUEST=, some of these are required, some optional, and some are invalid.	

The parameters are explained under the standard form of the RACROUTE macro instruction with the following exception:

,MF=(E, *ctrl addr)*
: specifies the execute form of the RACROUTE macro where *ctrl addr* is the address of the associated parameter list.

RACXTRT - RACF Extraction or Encryption

The RACXTRT macro instruction is used to retrieve certain specified fields from a RACF user profile or to encrypt certain clear-text (readable) data.

This macro instruction is only available to authorized callers (APF-authorized, in system key 0-7, or in supervisor state).

Note: Encryption and extraction are mutually exclusive.

Note: Only callers in 24-bit addressing mode can issue this macro. Callers executing in 31-bit addressing mode, who want to use the RACXTRT function, can code the RACROUTE macro.

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

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

TYPE=EXTRACT TYPE=ENCRYPT	
,ENTITY=*resource name addr*	*resource name addr:* A-type address, or register (2) - (12)
,RELEASE=*number*	*number:* 1.6 or 1.7 **Default:** RELEASE=1.6
If TYPE=EXTRACT is specified:	
,SUBPOOL=*subpool number*	*subpool number:* decimal digit, 0-255 **Default:** SUBPOOL=229
,FIELDS=*field addr*	*field addr:* A-type address or register (2) - (12)
If TYPE=ENCRYPT is specified:	
,ENCRYPT=*(data address,DES)* ,ENCRYPT=*(data address,HASH)* ,ENCRYPT=*(data address,INST)*	*data address:* A-type address or register (2) - (12)

Note: If TYPE=ENCRYPT is specified, the only other allowable parameters are ENTITY and ENCRYPT, with ENCRYPT being required.

The parameters are explained as follows:

TYPE=EXTRACT
specifies the function to be performed by the extract function routine.

Note: If TYPE=EXTRACT is specified, ENCRYPT= must not be coded.

If EXTRACT is specified, the operation performed is extraction of the userid, connect group and optionally, the encrypted password. The specified fields are retrieved from the profile whose name is given as the ENTITY value, in the USER class. If ENTITY is not specified, the current user's profile is used. Upon return, Register 1 contains the address of a result area that begins with a fullword containing the area's subpool and length.

The fields in the result area are in the order below:

Offset	Data	Length
0	subpool of area	1
1	length of area	3
4	offset to start of optional field to contain the encrypted password	2
6	reserved	18
24	specified or current user's default userid	8
32	specified user's default connect group or current user's current connect group	8

,SUBPOOL=subpool number
specifies the storage subpool from which the extract function routine will obtain an area needed for the extraction. If this parameter is not specified, it defaults to 229.

,FIELDS=address
specifies the address of a fullword with a value of binary 1 followed by an 8-byte character string, 'PASSWORD'. Use this parameter when the user's encrypted password is to be extracted in addition to the userid and user connect group. If FIELDS is specified, the encrypted password will be returned in the result area at an offset from the start of the area specified by the halfword at offset four.

TYPE=ENCRYPT
specifies the function to be performed by the extract function routine.

If TYPE=ENCRYPT is specified, the operation performed is data encryption. The ENCRYPT keyword specifies the data to be encrypted and the encryption method used. The first eight bytes of the area pointed to by the ENTITY operand will be used by the DES (Data Encryption Standard) encryption routine. If ENTITY is not specified, the userid from the current ACEE will be used instead. If TYPE=ENCRYPT is specified, no work area will be returned.

,ENCRYPT=(data address, DES)
,ENCRYPT=(data address, HASH)
,ENCRYPT=(data address, INST)
specifies the data to be encrypted, and a method of encryption. The address points to a one-byte length field followed by from 1 to 255 bytes of clear-text data to be encrypted. The second subparameter specifies the encryption method: the DES algorithm, the RACF hashing algorithm, or whatever scheme the installation uses (INST value). Upon return to the macro issuer, the first subparameter will now contain the address of an area

that contains a one-byte length followed by the encrypted version of the data. Neither the address itself nor the length is changed.

Note: When the DES algorithm is used RACF actually encrypts the data pointed to by the ENTITY profile, the userid, using the data as the encryption key. Data is one-way encrypted, that is, no facility is provided to recover the data in readable form. If HASH is specified, then the RACF hashing algorithm is used and data is masked instead of encrypted.

,ENTITY = resource name address
specifies the address of an eight-byte area containing the userid for which profile data is to be extracted, or the data to be used when encrypting. The data must be the RACF userid associated with the password or OID card data to be encrypted. The name must be left-justified in the field and padded with blanks. If this parameter is not specified, a default value of zero will indicate to RACF that the userid from the current ACEE will be used.

,RELEASE = *number*
specifies the RACF release level of the parameter list to be generated by this macro.

Certain parameters can be specified only with particular releases. If you specify a parameter with an incompatible release level, the parameter will not be accepted by the macro processing. An error message will be issued at assembly time. For the parameters that are valid for RELEASE = 1.6 and later, see Figure 12.

The default is RELEASE = 1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACXTRT macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

Parameters For RELEASE = 1.6 and Later

The RELEASE values for which a specific parameter is valid are marked with an 'X'.

Parameter	RELEASE = 1.6	RELEASE = 1.7
ENCRYPT =	X	X
ENTITY =	X	X
FIELDS =	X	X
SUBPOOL =	X	X
RELEASE =	X	X
TYPE =	X	X

Figure 12. RACXTRT Parameters for RELEASE = 1.6 and Later

Reason Codes and Return Codes

When control is returned, register 15 contains one of the following return codes and register 0 may contain a reason code.

Hexadecimal Code	Meaning
00	The extraction or encryption completed successfully.
04	An ESTAE environment was not able to be established, or if Register 0 contains a reason code of 1, neither EXTRACT nor ENCRYPT was specified for TYPE=.
08	For the extract function, the RACF user profile could not be found.
12	For TYPE=EXTRACT, RACF is inactive.
16	The extract operation failed. Register 0 contains the RACF manager return code which caused termination. This return code is not used for the encrypt function.
20	An ACEE address was not found when required, or if found, was not for a defined user.
24	A parameter list error was encountered.
64	Indicates that the CHECK subparameter of the RELEASE keyword was specified on the execute form of the RACXTRT macro; however, the list form of the macro does not have the proper RELEASE parameter. It also indicates that the TYPE parameters specified on the list and execute forms may not be the same TYPE. Macro processing terminates.

RACXTRT (List Form)

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

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

TYPE=EXTRACT TYPE=ENCRYPT	
,ENTITY=*resource name addr*	*resource name addr:* A-type address.
,RELEASE=*number*	*number:* 1.6 or 1.7 **Default:** RELEASE=1.6
,MF=L	
If TYPE=EXTRACT is specified:	
,SUBPOOL=*subpool number*	*subpool number:* decimal digit, 0-255 **Default:** SUBPOOL=229
,FIELDS=*field addr*	*field addr:* A-type address
If TYPE=ENCRYPT is specified:	
,ENCRYPT=*(data address,DES)* ,ENCRYPT=*(data address,HASH)* ,ENCRYPT=*(data address,INST)*	*data address:* A-type address

The parameters are explained under the standard form of the RACXTRT macro instruction with the following exception:

,MF=L
specifies the list form of the RACXTRT macro instruction.

RACXTRT (Execute Form)

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

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

TYPE=EXTRACT TYPE=ENCRYPT	
,ENTITY=*resource name addr*	*resource name addr:* RX-type address or register (2)-(12)
,RELEASE=*(number*,CHECK) ,RELEASE=*number* ,RELEASE=(,CHECK)	*number:* 1.6 or 1.7 **Default:** RELEASE=1.6
,MF=(E,ctrladdr)	crtl addr:RX-type address, register (1), or register (2)-(12)
If TYPE=EXTRACT is specified:	
,SUBPOOL=*subpool number*	*subpool number:* decimal digit, 0-255 **Default:** SUBPOOL=229
,FIELDS=*field addr*	*field addr:* RX-type address or register (2)-(12)
If TYPE=ENCRYPT is specified:	
,ENCRYPT=*(data address,DES)* ,ENCRYPT=*(data address,HASH)* ,ENCRYPT=*(data address,INST)*	*data address:* A-type address or register (2)-(12)

The parameters are explained under the standard form of the RACXTRT macro instruction with the following exception:

,RELEASE = (*number*,CHECK)
,RELEASE = *number*
,RELEASE = (,CHECK)
specifies the RACF release level of the parameter list to be generated by this macro.

Certain parameters can be specified only with particular releases. If you specify a parameter with an incompatible release level, the parameter will not be accepted by the macro processing. An error message will be issued at assembly time. For the parameters that are valid for RELEASE = 1.6 and later, see Figure 12 on page 2-327.

The default is RELEASE = 1.6.

When you specify the RELEASE keyword, checking is done at assembly time. Execution-time validation of the compatibility between the list and execute forms of the RACXTRT macro can be done by your specifying the CHECK subparameter on the execute form of the macro.

When CHECK processing is requested, if the size of the list-form expansion is not large enough to accommodate all parameters defined by the RELEASE keyword on the execute form of the macro, the execute form of the macro will not be done. Instead, a return code X'64' will be generated.

,MF = (E,ctrl addr)
specifies the execute form of the RACXTRT macro instruction using a remote control program parameter list.

RESERVE - Reserve a Device (Shared DASD)

The RESERVE macro instruction reserves 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 other systems. The reserve actually occurs when the first I/O is done to the device after the RESERVE macro is issued. When the reserving program no longer needs the reserved device, it should issue a DEQ macro instruction to release the resource. For information about how to obtain the UCB address for a device, see the section "Finding the UCB Address for the RESERVE Macro" in Volume 1.

If a task issues two RESERVE instructions for the same device without an intervening DEQ, an abnormal termination results unless the second RESERVE specifies the keyword parameter RET= or ECB=. (If a restart occurs when a RESERVE is in effect for resources, the system does not restore the RESERVE; the user's program must reissue the RESERVE.) If a DEQ is not issued for a particular resource, termination routines release resources reserved by a terminating task.

If global resource serialization is active, the hardware RESERVE can be suppressed leaving a SYSTEMS ENQ depending on the contents of the resource name lists. See *Planning: Global Resource Serialization* for information on resource name lists.

Global resource serialization counts and limits the number of concurrent resource requests in an address space. If an unconditional RESERVE (a RESERVE that uses the RET=NONE option) causes the count of global resource serialization requests to exceed the sum of a threshold value plus a tolerance value, an authorized caller is abended with a system code of X'538'. See "Limiting Global Resource Serialization Requests" in Volume 1.

To use shared DASD in higher level languages, write an assembler language subroutine to issue the RESERVE macro instruction. Pass the following information to this routine: ddname, qnameaddress, rnameaddress, rnamelength, and RET parameter. Refer to Figure "Example of Subroutine Issuing RESERVE and DEQ" in Volume 1 for additional information.

The ECB parameter is restricted in use to callers in supervisor state, PSW key 0-7, or with APF authorization. Except for the UCB, all input parameters to this macro instruction can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

A RESERVE used with the MASID and MTCB operands provides a special form of the RESERVE macro instruction that allows a further conditional control of a resource. One task, called the "issuing task" can issue a RESERVE macro instruction for a resource specifying the ASID and TCB of another task, called the "matching task." The MTCB and MASID operands are specified with RET=HAVE and ECB= to provide additional return codes. If the issuing task does not acquire control of the resource, it may receive a return code indicating that the resource is controlled by the matching task. Upon receiving this return code, the issuing task

could use the resource, if serialization between itself and the matching task has been accomplished by some pre-arranged protocol known to both the issuing and matching tasks.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RESERVE.
RESERVE	
b	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).
,	**Default:** E
,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:* A-type address, or register (2) - (12).
,UCB=*ucb addr*	*ucb addr:* A-type address, or register (2) - (12).
,MASID=*matching-asid addr*	*matching-asid addr:* A-type address, or register (2) - (12).
,MTCB=*matching-tcb addr*	*matching-tcb addr:* A-type address, or register (2) - (12).
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

(
: 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 among systems.

)

specifies the end of the resource description.

,RET=TEST
,RET=USE
,RET=HAVE

specifies a conditional request for all the resources named 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. If the return code for one or more requested resources is 4 and the request is not nullified by a corresponding DEQ, the ECB is posted when all the requested resources (specifically, those that initially received a return code of 4) are assigned to the requesting task.

,UCB=*ucb addr*

specifies the address of a fullword that contains the address of the UCB for the device to be reserved. The UCB must be allocated to the job step before RESERVE is issued unless the issuer is in supervisor state, system key, or APF-authorized.

Note: The UCB resides in storage below 16 megabytes.

,MASID = *matching-asid addr*

specifies the matching task (by defining a matching ASID) for the RESERVE, if used in conjunction with the MTCB parameter. MASID defines the ASID of a task that may be using a resource desired by the issuer of the RESERVE macro instruction.

Note: MASID can only be specified if MTCB is also specified.

,MTCB = *matching-tcb addr*

specifies the matching task (by defining a matching TCB) for the RESERVE, if used in conjunction with the MASID parameter. MTCB defines the TCB of a task that may be using a resource desired by the issuer of the RESERVE macro instruction.

If the task specified by the MASID and MTCB parameters is not using the resource, global resource serialization gives control to the issuer of the RESERVE and returns a return code indicating whether the resource can be used. If the task specified by MASID and MTCB parameters is using the resource, global resource serialization records a request for the resource, suspends the issuing task until the resource is available, or optionally returns a return code indicating that an ECB will be posted when the resource can be used.

The MASID and MTCB parameters are specified with RET=HAVE, RET=TEST, and/or ECB= parameters to elicit additional return codes that provide information about the owner of the resource.

Note: MTCB can only be specified if MASID is also specified.

,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.

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 code, as shown in Figure 13.

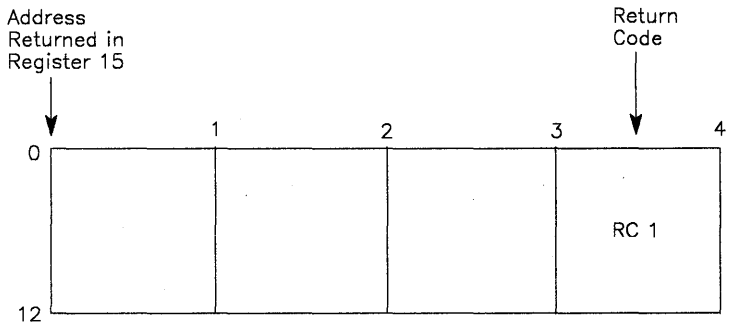


Figure 13. Return Code Area Used by RESERVE

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.
14	A previous request for control of the same resource has been made for the same task. Task does not have control of resource.
18	For RET=HAVE, RET=USE, or ECB=, the limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource. The ECB is not posted.
20	The matching task (the task specified in the MASID/MTCB parameters) owns the resource. The issuer of the RESERVE macro instruction may use the resource but it must ensure that the owning task does not terminate while the issuing task is using the resource. If the issuing task requested exclusive control then this return code indicates that the matching task is the only task that currently owns the resource. If the issuer of the RESERVE requested shared control and the owning task had requested shared control, this return code may indicate that a previous task had requested exclusive control. The issuing task must issue a DEQ to cancel this RESERVE. The ECB will not be posted.
24	The issuing task will have exclusive control after the ECB is posted. The issuing task may use the resource but must ensure that the matching task does not terminate while the issuing task is using the resource. The issuing task must issue a DEQ to cancel the RESERVE.
28	The issuing task cannot obtain exclusive control of the resource using the MASID/MTCB RESERVE. The matching task's involvement with other tasks precludes control by the issuing task. This task must not issue a DEQ to cancel the RESERVE. The ECB will not be posted.
44	The issuing task is violating a restriction of the MASID/MTCB RESERVE in one or more of the following ways: • Another task has already issued this RESERVE for this resource specifying the same MASID/MTCB. • The MASID/MTCB parameters specify a task that acquired control of the resource by using the MASID/MTCB RESERVE. • The matching task requested ownership of the resource but has not yet been granted ownership. The ECB will not be posted. Return code 44 is never given by a RESERVE RET=TEST, return code 4 is given instead.

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.
b	One or more blanks must precede RESERVE.
RESERVE	
b	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.
, ,SYSTEMS	
)	
,RET=TEST ,RET=USE ,RET=HAVE	
,ECB=*ecb addr*	*ecb addr:* A-type address.
,UCB=*ucb addr*	*ucb addr:* A-type address or 0.
,MASID=0 ,MTCB=0	
,RELATED=*value*	*value:* A-type address.
,MF=L	

The parameters are explained under the standard form of the RESERVE macro instruction, with the following exception:

,MF=L
specifies the list form of the RESERVE macro instruction.

Note: If the ECB parameter is specified on the execute form of the macro, it must also be specified on the list from of the macro. The list form of this macro generates a prefix followed by the parameter list, however the label specified in MF=L does not include an offset prefix area. If MASID, MTCB, TCB, or ECB are specified, these labels are offset; allowance must be made for the parameter list prefix.

RESERVE (Execute Form)

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RESERVE.
RESERVE	
b	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). **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:* RX-type address, or register (2) - (12).
,UCB=*ucb addr* ,MASID=*matching-asid addr* ,MTCB=*matching-tcb addr*	*ucb addr:* RX-type address, or register (2) - (12). *matching-asid addr:* A-type address, or register (2) - (12). *matching-tcb addr:* A-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) - (12).

The parameters are explained under the standard form of the RESERVE macro instruction, with the following exception:

,MF = (E,*ctrl addr***)**
specifies the execute form of the RESERVE macro instruction using a remote control program parameter list.

Note: If the ECB parameter is specified on the execute form of the macro, it must also be specified on the list form of the macro. If MASID and MTCB are specified, MASID = 0 and MTCB = 0 must be specified in the list form.

The list form of this macro generates a prefix followed by the parameter list, however the label specified in MF = L does not include an offset prefix area. If MASID, MTCB, TCB, or ECB are specified, these labels are offset; allowance must be made for the parameter list prefix.

RESUME - Resume Execution of a Suspended Request Block

The RESUME macro instruction causes suspended RBs to resume execution. RESUME is supported in cross memory mode. For restrictions on using the RESUME macro instruction, see "Resuming Execution of a Suspended Request Block" in Volume 1.

The RESUME macro instruction is coded as follows:

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

TCB=(4) TCB=*tcbaddr*	**Default:** TCB address contents of register (4). *tcbaddr:* A-type address or registers (2) - (12).
,RB=(5) ,RB=*rbaddr*	**Default:** RB address contents of register (5). *rbaddr:* A-type address or registers (2) - (12).
,RETURN=Y ,RETURN=N	**Default:** RETURN=Y
,MODE=UNCOND ,MODE=COND	**Default:** MODE=UNCOND.
,ASYNC=Y ,ASYNC=N	**Default:** ASYNC=N
,ASCB=*ascbaddr*	**Default:** ASCB address of the home address space. *ascbaddr:* RX-type address or registers (1) or (2) - (3) or (6) - (12).

The parameters are explained as follows:

TCB=(4)
TCB=*tcbaddr*
specifies the TCB address of the task to be resumed. Register (4) is the default; it is assumed to contain the TCB address.

Note: The TCB resides in storage below 16 megabytes.

,RB=(5)
,RB=*rbaddr*

specifies the address of the RB to be resumed. Register (5) is the default; it is assumed to contain the address of the RB to be resumed. The specification of the RB operand determines which RB will have its suspend count decremented (which RB will be made ready for resumption of execution).

Note: The RB resides in storage below 16 megabytes.

,RETURN=Y
,RETURN=N

specifies whether control is to be returned to the caller (RETURN=Y) or to the TCTL routine (RETURN=N). RETURN=N can only be issued if the caller is an SRB running in home mode that specifies MODE=UNCOND and ASYNC=N, and does not specify the ASCB option. The TCTL routine passes control directly from the SRB to a TCB.

,MODE=UNCOND
,MODE=COND

If MODE=COND is specified, the action RESUME takes if the function cannot be completed synchronously depends on the ASYNC option. If ASYNC=Y is specified, RESUME makes a conditional attempt to acquire an SRB from the supervisor SRB pool. If an SRB is available, it is scheduled to complete the RESUME function asynchronously. If ASYNC=N is specified explicitly or as a default, and the RESUME cannot immediately complete the function, it places return code 04 in register 15 and returns to the caller.

If MODE=UNCOND is specified, the action RESUME takes also depends on the ASYNC option. If ASYNC=Y is specified, RESUME makes an unconditional request for an SRB from the supervisor SRB pool, and completes the RESUME function asynchronously. If ASYNC=N is specified explicitly or as a default, RESUME unconditionally obtains the CML lock of the ASCB whose TCB or RB is to be resumed. The TCB or RB is resumed before control returns to the caller.

,ASYNC=Y
,ASYNC=N

specifies whether the RESUME is to be completed asynchronously (Y) or not (N).

Note: The ASYNC parameter for the RESUME macro instruction is spelled differently from similar parameters on other macro instructions.

,ASCB=*ascbaddr*

specifies the address of the ASCB whose TCB or RB is to be resumed. The caller must establish current addressability to the address space before calling RESUME. If this option is not specified, the home address space is assumed. This option must be specified if ASYNC=Y is specified.

Note: The ASCB resides in storage below 16 megabytes.

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

Hexadecimal Code	Meaning
00	A normal, synchronous RESUME completed the function.
04	For MODE=COND and ASYNC=N, the RESUME cannot complete the function. For MODE=COND or MODE=UNCOND and ASYNC=Y, an SRB is completing the function asynchronously.
08	For MODE=COND and ASYNC=Y the SRB pool is empty and RESUME cannot complete the function.

The RESUME macro instruction uses registers as follows:

Register	Use
0-1	Work registers
2-3	Unchanged
4	TCB address *
5	RB address *
6-10	Unchanged
11-13	Work registers
14	Return point
15	EPA of the RESUME routine, or return code, if RETURN=Y was specified.

* May be changed by the RESUME service.

Example 1

Operation: Resume execution of the task specified in the address labeled CURRTCB. Use the request block address in register 5. Pass control back to the task (the issuer is currently in SRB mode, and this step terminates SRB mode processing).

```
RESUME   TCB=CURRTCB,RB=(5),RETURN=N
```

RISGNL - Issue Remote Immediate Signal

The RISGNL macro instruction uses the emergency signal (EMS) order code of the signal processor (SIGP) instruction to invoke the execution of a specified software program on a specific processor in a multiprocessing configuration. The program may be requested to execute in parallel or serially with the function requesting the program. The specified software program (receiving routine) gets control disabled, in key 0, and supervisor state. If the routine that issues the RISGNL macro instruction holds the SALLOC lock, the receiving routine can enable for a malfunction alert or an emergency signal. Otherwise, the receiving routine cannot enable, request locks, or issue SVCs. In addition, the receiving routine must return via the address in register 14. The RISGNL macro instruction can be invoked by programs executing in cross memory mode.

Additional SIGP order codes are available via the DSGNL and RPSGNL macro instructions. See *Principles of Operation* for an explanation of the order codes.

The RISGNL macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RISGNL.
RISGNL	
b	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 as follows:

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 processor.

,CPU = *PCCA addr*
specifies the address of the physical configuration communication area (PCCA) of the processor on which the function is to be performed.

Note: The PCCA must reside in 24-bit addressable storage.

,EP = *entry name addr*
specifies the address of the receiving routine to be executed on the specified processor. The receiving routine will get control in the same addressing mode as the macro issuer.

,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 processor not online. If it appeared to be online, it is no longer in the configuration.
14	Processor alive bit was turned off during the remote immediate window spin routine.

Example 1

Operation: The routine whose address is in register 12 is to be given control on the processor whose PCCA address is in register 1. Return control to the caller when the specified receiving routine has been given control.

```
RISGNL PARALLEL,CPU=(1),EP=(12)
```

Example 2

Operation: The routine whose address is in register 12 is to be given control on the processor whose PCCA address is in register 1. The routine will complete before the caller of RISGNL receives control again. Register 11 contains the address of a parameter to be passed to the receiving routine.

```
RISGNL SERIAL,CPU=(1),EP=(12),PARM=(11)
```

RPSGNL - Issue Remote Pendable Signal

The RPSGNL macro instruction uses the external call (EC) order code of the signal processor (SIGP) instruction to invoke the execution of one of five software programs on a specific processor in a multiprocessing configuration.

Additional SIGP order codes are available via the DSGNL and RISGNL macro instructions. See *Principles of Operation* for an explanation of the order codes.

The RPSGNL macro instruction can be used by programs executing in cross memory mode.

The RPSGNL macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede RPSGNL.
RPSGNL	
b	One or more blanks must follow RSPGNL.

SWITCH RQCHECK GTFCRM MODE MEMSWT	
,CPU = *PCCA addr*	*PCCA addr:* RX-type address, or register (1).

The parameters are explained as follows:

SWITCH
RQCHECK
GTFCRM
MODE
MEMSWT
specifies the action to be performed:

SWITCH	Memory/task switch 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	Machine check handler (MCH) function, to modify MCH-oriented control registers
MEMSWT	Memory switch function, to update the PSAANEW on the specified processor.

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

Note: The PCCA must reside in 24-bit addressable storage

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

Hexadecimal Code	Meaning
00	Specified processor is online and has been notified that the specified service is to be executed.
04	Function not initiated because addressed processor not online. If it appeared to be online, it is no longer in the configuration.

Example 1

Operation: Interrupt the work running on the processor whose PCCA address is given in register 1. If the work is preemptable, pass control to the dispatcher.

```
RPSGNL SWITCH,CPU=(1)
```

Example 2

Operation: Pass control to the memory switch function on the processor whose PCCA address is given in register 1.

```
RPSGNL MEMSWT,CPU=(1)
```

SCHEDULE - Schedule System Services for Asynchronous Execution

If SCOPE=GLOBAL is coded, this macro can be assembled compatibly between MVS/XA and MVS/370 through the use of the SPLEVEL macro instruction. Default processing will result in an expansion of the macro that operates only with MVS/XA. The expansion of SCHEDULE with SCOPE=LOCAL will operate on both MVS/370 and MVS/XA. See the topic "Selecting the Macro Level" for additional information.

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 have a priority that is greater than, and independent of, any address space priority. Services scheduled at a local priority have the priority of the specific address space they execute in, but still have a priority greater than that of any task within the address space. To use SCHEDULE you must be in supervisor state, PSW key zero.

The addressing mode of the SRB routine is specified in the SRBEP field of the SRB control block. The user is required to set the correct AMODE. If bit 0 of the SRBEP field is set to 1, the SRB gets control in 31-bit addressing mode; if bit 0 is set to 0, the SRB routine gets control in 24-bit addressing mode. The addressing mode of the SRB's FRR is specified in the SRBFRRA field of the SRB control block. The user is required to set the correct AMODE. If bit 0 of the SRBFRRA field is set to 1, the FRR (and its retry routine) get control in 31-bit addressing mode. If bit 0 of the SRBFRRA field is set to 0, the FRR (and its retry routine) get control in 24-bit addressing mode.

The SCHEDULE macro instruction can be used by programs executing in cross memory mode. The SCHEDULE macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede SCHEDULE.
SCHEDULE	
b	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
,LLOCK=YES ,LLOCK=NO	**Default:** LLOCK=NO
,FRR=YES ,FRR=NO	**Default:** FRR=NO
,DISABLED	

The parameters are explained as follows:

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.

,LLOCK=YES
,LLOCK=NO
specifies whether the SRB is to receive control with the LOCAL lock held.

Note: CML (cross memory local) lock means the local lock of an address space other than the home address space. LOCAL lock means the local lock of the home address space. When written in lower case, local lock means any local-level lock, either the LOCAL or a CML lock.

,FRR=YES
,FRR=NO
specifies whether the SRB is to receive control with recovery established. If FRR=YES is specified, the user must include in the SRB field (SRBFRRA) the FRR exit address. When the SRB receives control, the FRR will have been added to the FRR stack. When FRR=YES is specified, the 24 byte FRR parameter area address will be passed to the SRB routine in register 2.

,DISABLED
specifies the mode of the issuer (if known) when the SCHEDULE macro is executed. DISABLED should be specified only when the invoker is physically disabled for interrupts.

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
```

Example 3

Operation: Schedule an SRB at a global priority specifying that the SRB is to receive control with the LOCAL lock held and recovery established. The issuer of the SCHEDULE macro instruction is disabled.

```
SCHEDULE SRB=(1),SCOPE=GLOBAL,LLOCK=YES,FRR=YES,DISABLED
```

SDUMP - Dump Virtual Storage

This macro can be assembled compatibly between MVS/XA and MVS/370 through the use of the SPLEVEL macro instruction. Default processing and the use of one of the new MVS/XA parameters will result in an expansion of the macro that operates only with MVS/XA. The expansion of SDUMP without new parameters will operate on both an MVS/XA and MVS/370 systems. See the topic "Selecting the Macro Level" for additional information.

Except for the DCB, all input parameters to this macro instruction can reside in storage above 16 megabytes if the caller is executing in 31-bit addressing mode.

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 zero, supervisor state, and must be in SRB mode, or own a lock, or be disabled (with supervisor bit on) or be in enabled unlocked task FRR mode with the corresponding PSA bit on.

The issuer of SDUMP can be in any cross memory mode.

The service of initiating an SVC Dump in any address space is provided for callers who need to dump address spaces 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.

When SVC dump is entered, the specified parameter list and all areas the list points to (except the DCB and ECB) must be currently addressable. Both the DCB and ECB, if specified, are assumed to be addressable in the home address space.

The system operator can take SVC Dumps by issuing the DUMP command. For more information see *Operations: System Commands.*

If options requiring address constants (ACONs) are not specified, the standard form of the SDUMP macro instruction produces reentrant code for routines that reside in the link pack area. The following parameters do not require ACONs and produce reentrant code for routines that reside in the link pack area:

HDRAD
SDATA
TYPE
HDR
BRANCH
SUSPEND
QUIESCE
BUFFER

SVC dump allows programs in page-protected storage (such as the nucleus, PLPA, and MLPA) to issue the standard form of the SDUMP macro instruction without causing a protection exception.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede SDUMP.
SDUMP	
b	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).
,ASIDLST=*list addr*	*list addr:* RX-type address, or register (2) - (12).
,TYPE=*(type code)*	*type code:* any of the following, separated by commas: XMEM, XMEME, NOLOCAL, FAILRC **Note:** XMEM and XMEME are mutually exclusive codes.
,ECB=*ecb addr*	*ecb addr:* A-type address, or register (2) - (12). **Note:** If ECB is specified, ASID or ASIDLST must also be specified.
,SDATA=*(data code)*	*data code:* any combination of the following, separated by commas: ALLNUC, ALLPSA, CSA, GRSQ, LPA, LSQA, NOALLPSA/NOALL, NOSQA, NOSUMDUMP/NOSUM, NUC, PSA, RGN, SQA, SUMDUMP/SUM, SWA, TRT **Notes:** 1. Executing the SDUMP macro results in the ALLPSA, SQA, and SUMDUMP storage areas being dumped unless excluded by the NOALLPSA, NOSQA, or NOSUMDUMP parameter. 2. The PSA option is not required because it is dumped as a default in all SVC dumps.
,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)*
,LISTA=*list addr*	*list addr:* A-type address or register (2) - (12).
,SUBPLST=*subpool id list addr*	*subpool id list addr:* RX-type address, or register (2) - (12).
,KEYLIST=*storage key list addr*	*storage key list addr:* RX-type address, or register (2) - (12). **Note:** KEYLIST cannot be specified without SUBPLST.
,BUFFER=NO ,BUFFER=YES	**Default:** BUFFER=NO
,QUIESCE=YES ,QUIESCE=NO	**Default:** QUIESCE=YES
,BRANCH=NO ,BRANCH=YES	**Default:** BRANCH=NO **Note:** If BRANCH=YES is specified, ASID or ASIDLST must also be specified.
,SUSPEND=NO ,SUSPEND=YES	**Default:** SUSPEND=NO
,SUMLIST=*list addr*	*list addr:* RX-type address, or register (2) - (12).
,SUMLSTA=*list addr*	*list addr:* RX-type address or register (2) - (12).

The parameters are explained as follows:

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 be 1-100 characters 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.

If these keywords are specified with BRANCH=YES or ASID/ASIDLST (that is, causing a scheduled dump), the title is moved to SVC dump storage before control returns to the caller. There is no requirement to synchronize with the completion of the dump.

,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 is used. The data control block must be addressable from all the address spaces in which the SVC Dump routine executes. The control blocks built by OPEN must also be addressable from the address spaces. The DCB must support EXCP.

The DCB must reference device types supported by SVC dump. Eligible device types are unlabeled 9-track 2400-series tape devices (or tape devices compatible with the 2400-series) and any direct access devices supported by the system that have a track size of at least 4104 bytes. (4104 bytes equals 1 SVC dump output record.) The IBM 3850 Mass Storage System is not supported as a dump data set.

SVC dump does not close the dump data set. The caller should use the CLOSE macro to close the data set and cause an end-of-file mark or a tape mark to be placed after the dump data. SVC dump sets up the DCB so that CLOSE works correctly and positions the end-of-file mark or tape mark at the correct place on the data set. For tape data sets, the caller can write a tape mark to separate multiple dumps without using the CLOSE macro.

Note: The DCB resides in storage below 16 megabytes.

,ASID = *ASID addr*
,ASIDLST = *list addr*
specifies the address of a halfword or a list of halfwords containing the hexadecimal address space identifier of an address space to be dumped. If register notation is used, the low order halfword of the register contains the address space identifier of the address space to be dumped. If both parameters are omitted, the current address space will be dumped. If 0 is specified for the address space identifier, a dump is scheduled for the home address space of the issuer of the SDUMP macro instruction. No private area storage will be included in the dump for the specified address space(s) if either of the following events occurred:

- No SDATA parameters were specified that apply to the private area of the requested address space(s).

- The CHNGDUMP operator command was used to set an overriding parameter in the system dump options list that limits SVC dumps to areas outside of the private area.

The ASID list can contain a maximum of 15 address space identifiers. The high order bit of the halfword containing the last identifier of the list must be set to 1, and all other high order bits must be set to 0.

,TYPE=XMEM
,TYPE=XMEME
,TYPE=NOLOCAL
,TYPE=FAILRC

specifies that the caller's cross memory mode is to be used to decide the address spaces to dump (XMEM or XMEME) or that the caller cannot allow SDUMP to obtain a local lock (NOLOCAL) or that SVC DUMP should return a reason code with the return code to the DUMP command processor when the requested dump was not taken (FAILRC).

XMEM requests SVC dump to use the caller's cross memory mode at the time the SDUMP macro instruction is executed.

XMEME requests SVC dump to use the caller's cross memory mode at the time of the error for which the dump is being taken.

The home address space is dumped for both keywords. The relevant primary and secondary address spaces are also dumped if they are unique. If a cross memory local lock was held, the address space whose local lock is held is also dumped.

NOLOCAL indicates that the caller is in an environment that cannot tolerate SDUMP obtaining a local lock. This option has meaning only when BRANCH=YES is specified and the caller is enabled and unlocked (for example, in an enabled unlocked task FRR or in SRB or cross memory mode).

FAILRC requests the system to return the explanation for any failures in the dump function. When you specify this parameter, SVC DUMP passes back a return code in register 15 and places the reason code in the SDWA. The reason code explains why the dump failed.

,ECB=*ecb addr*

If an A-type operand is specified, *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 a register operand is used, the register must contain the actual address of the event control block. If this parameter is omitted, the caller is not notified of the completion of the scheduled dump. The fullword and the event control block must be addressable from the home address space. The fullword address that points to the event control block must be a valid 24-bit or 31-bit address.

Note: The ECB will be posted only if the return code from SDUMP is 0.

,SDATA=*(data code)*

specifies the system control program information to be dumped:

ALLNUC -	The DAT-ON and DAT-OFF nuclei. The read only (page-protected) area of the nucleus and the DAT-OFF nucleus will not be included in the dump unless this keyword is specified.
ALLPSA -	All of the prefixed storage areas in the system.
CSA -	The common service area subpools (subpools 231 and 241).
GRSQ -	Global resource serialization control blocks are included in the dump.
LPA -	The active link pack area modules and SVCs for each address space being dumped.
LSQA -	The local system queue area for each address space being dumped (subpools 229, 230, 233-235, and 253-255).

NOALLPSA(NOALL) - The prefixed storage area for one processor is dumped. This is either the processor at the time of the error or the processor at the time of the dump.

NOSQA - The system queue area is not dumped.

NOSUMDUMP(NOSUM) - A summary dump is not included in the SVC Dump.

NUC - The non-page protected areas of the DAT-ON nucleus. (The ALLNUC parameter must be specified to obtain the entire nucleus, including the page-protected areas of the DAT-ON nucleus and the DAT-OFF nucleus.)

PSA - The prefixed storage area for one processor is dumped. This is either the processor at the time of the error or the processor at the time of the dump.

RGN - The allocated pages in the private area of each address space being dumped. This includes the following areas:

Subpools	Storage
0-127, 251, 252	All virtual storage in the address space allocated to these subpools, that resides below and above the 16 megabytes line
229, 230, 253-255	All virtual storage allocated to the LSQA and ELSQA
236 and 237	All virtual storage allocated to the SWA and ESWA

SVC DUMP does not dump all the obtained storage in an address space if the RGN option of SDATA is specified. This reduces the number of page faults that occur during SVC DUMP processing, decreases the time required to take a dump, and reduces the size of dumps on tape or DASD.

For storage that is not related to data-in-virtual, only obtained pages that have something stored into them are dumped. This eliminates the pages of storage that are in a freshly obtained state.

For storage that is related to data-in-virtual, pages that are in real storage are dumped, as well as pages that have been changed since the last DIV macro (that specified the SAVE service) executed.

When the RGN option of SDATA is coded on the SDUMP macro, SVC DUMP uses the RSM facility, IARQDUMP, to determine which category a page returned by VSMLIST belongs to. The categories are:

1. A copy of the page cannot be found in virtual storage.
2. All copies in virtual storage are at the same level as the copy on permanent storage.
3. At least one copy of the page has been found in virtual storage that is at a later level than the copy on permanent storage.
4. RSM cannot determine the status of the page.

SVC DUMP does not dump pages in the first category. It dumps pages in category 2 that are in real storage and all of category 3. If the page is in category 4, SVC DUMP references the page and then calls IARQDUMP again. If the page is still in category 4, it is not dumped. If the page is no longer in category 4, it is treated like any other page in its category.

SQA - The entire system queue area.

SUMDUMP(SUM) - A summary dump is to be included with the SVC dump output. The trace table is included in the non-summary portion of the dump if this option is specified or used as a default.

SWA - The scheduler work area subpools for each address space being dumped (subpools 236 and 237). This includes all virtual storage allocated above and below the 16 megabytes line.

TRT - The system trace table, the GTF trace records, and MASTER TRACE DATA if these types of traces are active.

If the BRANCH=YES and the SUSPEND=NO parameters are specified, the following system areas are included in the summary dump output:

- A cross memory status record indicating the home, primary, and secondary address spaces at the time SVC dump is entered. If the caller held a cross memory local lock, the ASCB address of the address space whose local lock is held is also displayed.
- Any storage areas specified with the SUMLIST or SUMLSTA parameters.
- The physical configuration communication area (PCCA), the logical configuration communication area (LCCA), and the prefixed storage area (PSA) for each functioning processor.
- The current PCLINK stack elements pointed to by PSASEL.
- The relevant interrupt handler save area (IHSA). If no local lock is held or the local lock of the home address space is held, the relevant IHSA is located in the home address space. If a cross memory local lock is held, the relevant IHSA is located in the address space whose local lock is held.
- 4K of storage before and 4K after the address in the PSW and every valid unique address in the registers that are saved in the relevant IHSA. This storage is dumped using the primary and secondary address space addressability of the unit of work whose status is saved in the relevant IHSA.
- The XSB associated with the relevant IHSA and the PCLINK stack elements pointed to by this XSB.
- The system diagnostic work area (SDWA) associated with the failure of the system routine.
- 4K of storage before and 4K after every valid unique address in the registers that are saved in this SDWA. This storage is dumped using the primary and secondary address space addressability of the process encountering the error.
- The global, CPU, and local work/save area vector tables (WSAVTG, WSAVTC, WSAVTL) and the work/save areas pointed to by the addresses in these vector tables.
- 4K of storage before and 4K after the instruction counter values of the external old PSW, program check old PSW, I/O old PSW, and restart old PSW saved in the PSA of all active processors.
- The functional recovery routine (FRR) stack for the current processor.

If the BRANCH=YES and SUSPEND=YES parameters are specified, the following system areas are included in the summary dump output:

- A cross memory status record indicating the home, primary, and secondary address spaces at the time SVC dump is entered. If a cross memory local lock is held, the ASCB of the address space whose local lock is held is also displayed.
- Any storage areas specified with the SUMLIST or SUMLSTA parameters.

- The physical configuration communication area (PCCA), the logical configuration communication area (LCCA), and the prefixed storage area (PSA) for each functioning processor.

- The interrupt handler save area relevant to the suspended SDUMP caller. If no local lock is held or the local lock of the home address space is held, the relevant IHSA is located in the home address space. If a cross memory local lock is held, the relevant IHSA is located in the address space whose local lock is held.

- The XSB pointed to by the relevant IHSA.

- The PCLINK stack elements pointed to by this XSB.

- The ASCB of the SVC dump caller.

- The suspended unit of work: the TCB/RB/XSB for task mode callers and the SSRB/XSB for SRB mode callers.

- The PCLINK stack elements associated with the suspended unit of work.

- For task mode SVC dump callers, the following areas are also dumped:

  All RTM2WAs associated with the suspended caller's TCB.
  All SDWAs pointed to by the located RTM2WAs.

- For SRB mode callers, the following areas are also dumped:

  The PCLINK stack elements pointed to by the SSRB/XSB.
  The SDWA located as the one for which the dump is being taken.

- The SVC dump caller's register save area.

- 4K of storage before and 4K after the unique register values found in all located SDWAs and the caller's register save area. This storage is dumped using the primary and secondary addressability of the program whose registers were saved in these system areas.

- 4K of storage before and 4K after the address portion of the PSWs in all located SDWAs.

If the BRANCH=NO parameter is in effect, the following system areas are included in the summary dump output:

- The ASID record for the address space in which the dump task is executing.

- Any storage areas specified with the SUMLIST or SUMLSTA parameter.

- Every RTM2 work area (RTM2WA) associated with every TCB in a dumped address space.

- 4K of storage before and 4K after all valid, unique PSW and register addresses saved in each RTM2WA. This storage is dumped using the primary and secondary addressability saved in the RTM2WA containing the PSW and register values.

In addition to a total of three pages around the PSW address at the time of the error, the LPA directory and the nucleus map, the following system control blocks are dumped in all SVC Dumps:

- The communications vector table (CVT), the CVT prefix, and the secondary CVT (SCVT)
- The global data area (GDA)
- The prefixed storage area (PSA) for the current processor
- Unit control blocks (UCBs)
- The address space vector table (ASVT)
- The address space control block (ASCB) for each address space being dumped
- The linkage table, authorization table, and entry tables for each requested address space
- The PCCA vector table and all PCCAs
- The trace option block (TOB)
- The trace vector table (TRVT)

Executing the SDUMP macro results in the ALLPSA, SQA, and SUMDUMP storage areas being dumped unless excluded by the NOALLPSA, NOSQA, or NOSUMDUMP parameter.

The system dump options specified by the CHNGDUMP operator command can add to or override the SDATA options specified with the SDUMP macro instruction.

,STORAGE= *(strt addr,end addr)*
,LIST= *list addr*
,LISTA= *listaddr*

specifies one or more pairs of starting and ending addresses (STORAGE), a list of starting and ending addresses (LIST), or a list of ASIDs and storage ranges (LISTA).

(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.

LISTA specifies a list of storage ranges as follows:

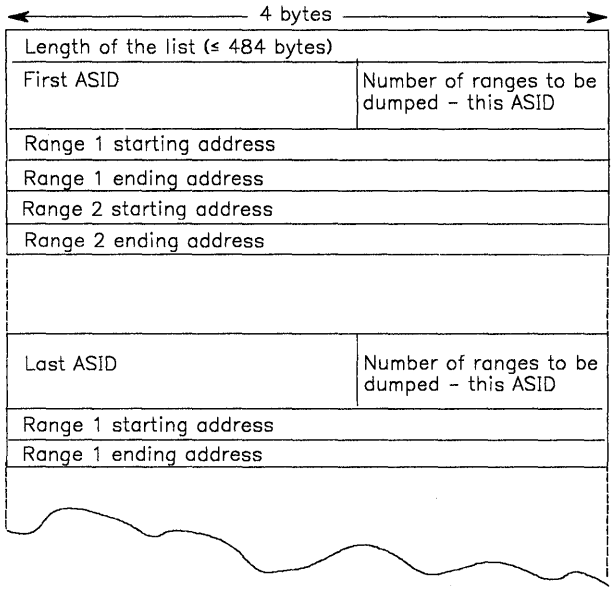


Figure 14. List of Storage Ranges Specified by LISTA

Note: If LISTA or SUBPLST is specified for a scheduled dump request and if the list does not exceed 484 bytes in size, SVC dump will move the list to SVC dump storage. The caller can free or reuse this space on return from SVC dump. No synchronization with SVC dump completion is required. If the list exceeds 484 bytes, SVC dump will not move the list and synchronization with SVC dump completion is required.

,SUBPLST=*subpool id list address*
specifies a list of ASIDs with associated subpool ids corresponding to subpools of virtual storage that are to be included in the SVC dump.

The first fullword of the list contains the number of bytes (including the first word) in the list. The list can contain a maximum of 200 bytes consisting of unique ASIDs and subpool ids. If the list contains duplicate ASIDs or subpool ids, the length can exceed 200 bytes because SDUMP stores the unique subpool ids in a 200-byte work area.

The structure of the list for each ASID follows:

- The first word contains the ASID in bits 0-15 and the number of subpools associated with this ASID (n) in bits 16-31. If 0 is specified as the ASID, the caller's home ASID is used.

- The next n halfwords contain the subpool ids (right justified) corresponding to the virtual storage to be included in the SVC dump. The manner in which these subpools are dumped depends on whether they are private or common area subpools.

  - If a private area subpool (related to a TCB) is specified, all virtual storage associated with this subpool, for all TCBs in the specified address space, is dumped.

  - If a common area subpool is specified, all of the virtual storage allocated in the subpool is dumped.

SVC DUMP does not dump all the obtained storage in an address space if the SUBPLST list keyword for private subpools is coded. This reduces the number of page faults that occur during SVC DUMP processing and the time required to take a dump. It also reduces the size of dumps on tape or DASD.

For storage that is not related to data-in-virtual, only obtained pages that have something stored into them are dumped. This eliminates the pages of storage that are in a freshly obtained state.

For storage that is related to data-in-virtual, only pages that are in real storage are dumped, as well as pages that have been changed since the last data-in-virtual SAVE operation.

When SUBPLST for private subpools is coded, SVC DUMP uses the RSM facility, IARQDUMP, to determine which category a page returned by VSMLIST belongs to. The categories are:

1. A copy of the page cannot be found in virtual storage.

2. All copies in virtual storage are at the same level as the copy on permanent storage.

3. At least one copy of the page has been found in virtual storage that is at a later level than the copy on permanent storage.

4. RSM cannot determine the status of the page.

SVC DUMP does not dump pages in the first category. It dumps pages in category 2 that are in real storage and all of category 3. If the page falls into category 4, SVC DUMP references the page and then calls IARQDUMP again. If the page is still in category 4, it is not dumped. If the page is no longer in category 4, it is treated like other page in its category.

Notes:

1. *If the KEYLIST option is specified, only the storage with keys matching the keys in the list is dumped.*

2. *Unassigned subpool ids and ASIDs are skipped.*

3. *If an invalid subpool or ASID (ASID greater than ASVTMAX) is specified, the caller receives a 233 ABEND and SDUMP processing terminates the dump.*

4. *If all ASIDs specified in SUBPLST are the current ASID, SUBPLST does not force a scheduled dump. However, if any of the ASIDs are different, a scheduled (or asynchronous) dump results.*

5. *SDUMP callers executing in key 0 and supervisor state, who request storage from subpool 0 via GETMAIN obtain that storage from subpool 252 instead. Therefore, when these callers want to dump this storage, they must specify subpool 252 rather than subpool 0.*

,KEYLIST=*storage key list addr*

specifies the address of a list of storage keys associated with the virtual storage to be dumped. If the key of a subpool specified in SUBPLST does not match a key in this list, the data in the supbool is not dumped. SUBPLST must be specified if the KEYLIST option is used. If KEYLIST is not specified, all virtual storage (regardless of key) associated with the requested subpools will be included in the dump.

The list contains one-byte entries and starts on a halfword boundary. The first byte indicates the length of the list (including this byte). The list has a maximum length of 16 bytes so that up to 15 keys can be specified. Callers should specify each key in the leftmost four bits of each byte, except the length byte.

Callers who want to dump the storage corresponding to all 16 keys should not specify this parameter.

,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.) Using BUFFER=YES requires special serialization. Refer to the topic "SQA Buffer Option" in Volume 1.

,QUIESCE=YES
,QUIESCE=NO

specifies that the system is to be set non-dispatchable until the contents of the SQA and the CSA are dumped (YES), or that the system is to be left dispatchable (NO). If the SDATA parameter does not specify SQA or CSA, the QUIESCE=YES request is ignored.

Note: Summary dumps (SUMDUMP) for branch entries (BRANCH=YES) always cause the system to be set non-dispatchable until the summary dump is written.

,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 (NO). For BRANCH=NO, the caller cannot be in cross memory mode. For BRANCH=YES, the caller can be in either cross memory mode or non-cross memory mode and must be in PSW key 0, supervisor state, and one of the following:

SRB mode
Holding any lock
Disabled with a PSASUPER bit on
Enabled unlocked task FRR mode with the corresponding PSA bit on

If BRANCH=YES is specified and the caller has not specified at least one of the following keywords: ASID, ASIDLST, TYPE=XMEM, TYPE=XMEME, or LISTA, the dump is scheduled to the current home address space.

Routines that issue SDUMP with BRANCH=YES must provide a 72-byte save area pointed to by register 13.

For BRANCH=YES entry by reentrant recovery routines, SDUMP processing moves the data supplied by the following parameters to a system area:

HDR
HDRAD
ASIDLIST
STORAGE
LIST
LISTA
SUBPLST
KEYLIST

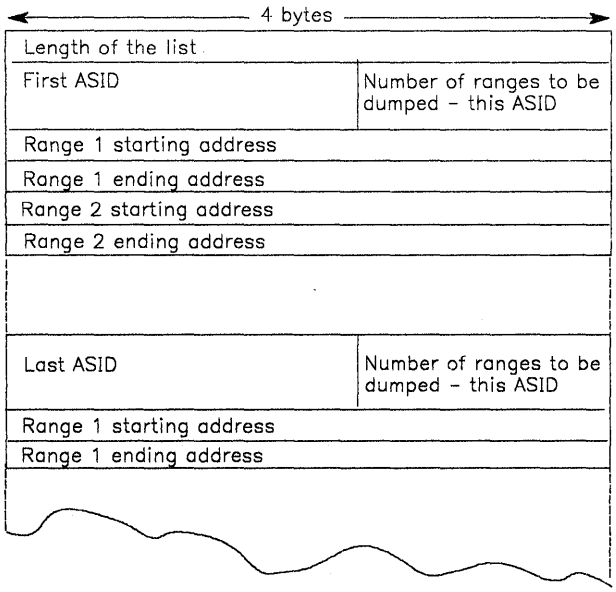
This enables the recovery routine to free its storage on return from SDUMP although the dump has not completed.

,SUSPEND=NO
,SUSPEND=YES
specifies that a suspend summary dump is requested (YES) or not requested (NO). SUSPEND=YES must be used together with the BRANCH=YES and SDATA=SUMDUMP parameters. This keyword should be used by system routines that can experience page faults but that want to save volatile system dump information in a virtual storage buffer.

,SUMLIST=*list addr*
,SUMLSTA=*list addr*
specifies a list of starting and ending addresses of areas to be included in a summary dump (SUMLIST) or specifies a combined list of ASIDs and storage ranges (SUMLSTA). For SUMLIST, SUMDUMP must be specified as an SDATA parameter and each starting address must be less than its corresponding ending address when either keyword is specified.

For SUMLIST, 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, and all other high order bits must be set to 0.

For SUMLSTA, the list of storage ranges is as follows:



Restriction:

- The maximum number of ASIDs that the combined TYPE=XMEM, TYPE=XMEME, LISTA, ASIDLST, ASID, and SUBPLST parameters can specify is fifteen.

  Note: There is no restriction on the number of ASIDs that the SUMLSTA can specify.

When BRANCH=YES and SUSPEND=NO are also specified, the list must be addressable using the addressability established when SVC dump was entered. The lists themselves and all ranges specified must reference paged-in data. Paged-out data is not dumped by summary dump.

When BRANCH=YES and SUSPEND=YES are also specified, the lists must be addressable using the addressability established when SVC dump was entered. The lists and referenced data can either be in paged in or paged out areas. The maximum amount of summary dump data with this type of dump is 1M.

When BRANCH=NO is also specified, the lists must be addressable in all address spaces in which the dump will be taken (those address spaces specified by ASID, ASIDLST, LISTA, or TYPE=XMEM, TYPE=XMEME, or SUBPLST). The lists and referenced data can be in paged-in or paged-out areas. The maximum amount of summary dump data possible with this type of dump is dependent only on the size of the dump data set.

Each ASID specified with SUMLSTA must represent a valid, swapped-in address space in order for the data to be dumped.

Programming Notes:

The total number of distinct ASIDs that can be specified by TYPE=XMEM, TYPE=XMEME, LISTA, ASID, SUBPLST and ASIDLST is fifteen. If more than fifteen are requested, only the first fifteen are processed. There is no restriction on the number of ASIDs specified by the SUMLSTA parameter, nor do SUMLSTA ASIDs contribute toward the fifteen ASID limit.

If BRANCH=NO was specified and no ASIDs other than the current ASID were requested, 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 because the dump data set did not have sufficient space.
08	The system was unable to take a dump.

If BRANCH=YES or any ASID other than the current ASID was requested, 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 in the ECB:

Hexadecimal Code	Meaning
00	A complete dump was taken.
04	A partial dump was taken.
08	The system was unable to take a dump.

Notes:

1. *When a return code of 8 is received, the SDWA (SDWASDRC) contains the reason why no dump was taken. See "Determining the Initial Status of an SVC Dump Request" in Volume 1 for a list of reasons.*

2. *The ECB will not be posted unless the return code from SDUMP is 0.*

Example 1

Operation: This example shows how SVC DUMP can be branch entered to initiate a dump in an address space by callers who cannot issue an SVC. Areas to be dumped are requested via three parameters (BUFFER, SDATA, and STORAGE). The dump has the title indicated in the HDR parameter, the caller requests to be notified of the completion of the scheduled dump via the ECB parameter, and the dump is going to a private data set (indicated by the DCB option).

```
SDUMP HDR='USER DATA FOR TEST A',DCB=TESTADCB,BUFFER=YES,   X
  ASID=TSTAASID,ECB=(8),QUIESCE=YES,BRANCH=YES,            X
  STORAGE=(A,B,C,D,(9),E),SDATA=(ALLPSA,SQA,LSQA)
```

Example 2

Operation: This example shows how SVC DUMP can be invoked via a branch entry to initiate a dump of several address spaces by callers who cannot issue an SVC. Areas to be dumped are requested via four parameters (BUFFER, SDATA, LIST, and SUMLIST). The address spaces to be dumped are described by the ASIDLST parameter. Note that areas specified by SUMLIST only apply to the current address space. The LIST addressed by the LIST keyword must be addressable from any address space. The dump has 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 B',                              X
  BUFFER=YES,ASIDLST=TSTALIST,ECB=(8),                         X
  QUIESCE=YES,BRANCH=YES,LIST=(9),                             X
  SDATA=(ALLPSA,NUC,SQA,SUMDUMP),                              X
  SUMLIST=TSTSLIST
  .
  .
  .
TSTALIST DC X'0000000A800B'
TSTSLIST DC X'0000000080400000'
```

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.
b	One or more blanks must precede SDUMP.
SDUMP	
b	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: ALLNUC, ALLPSA, CSA, GRSQ, LPA, LSQA, NOALLPSA/NOALL, NOSQA, NOSUMDUMP/NOSUM, NUC, PSA, RGN, SQA, SUMDUMP/SUM, SWA, TRT **Default:** SDATA = (ALLPSA,SQA,SUMDUMP) ALLPSA, SQA, and SUMDUMP are the defaults even if other parameters are specified for SDATA. NOALLPSA, NOSQA, and NOSUMDUMP must be specified to suppress these defaults. **Note:** The PSA option is not required because it is dumped as a default in all SVC dumps.
,STORAGE = *(strt addr,end addr)* ,LIST = *list addr* ,LISTA = *list addr*	*strt addr:* A-type address. *end addr:* A-type address. *list 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)*
,SUBPLST = *subpool id list addr*	*subpool id list addr:* A-type address, or register (2) - (12).
,KEYLIST = *storage key list addr*	*storage key list addr:* A-type address, or register (2) - (12). **Note:** KEYLIST cannot be specified without SUBPLST.
,BUFFER = NO ,BUFFER = YES	**Default:** BUFFER = NO
,QUIESCE = YES ,QUIESCE = NO	**Default:** QUIESCE = YES
,SUSPEND = NO ,SUSPEND = YES	**Default:** SUSPEND = NO
,TYPE = *(type code)*	*type code:* any combination of the following, separated by commas: XMEM or XMEME, NOLOCAL
,MF = L	

The parameters are explained under the standard form of the SDUMP macro instruction, with the following exception:

,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 are lost.

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

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

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).
,ASIDLST=*list addr*	*list addr:* RX-type address, or register (2) - (12).
,TYPE=*(type code)*	*type code:* any one of the following, separated by commas: XMEM or XMEME, NOLOCAL
,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: ALLNUC NOSUMDUMP/NOSUM ALLPSA NUC CSA PSA GRSQ RGN LPA SQA LSQA SUMDUMP/SUM NOALLPSA/NOALL SWA NOSQA TRT **Default:** SDATA=(ALLPSA,SQA,SUMDUMP) ALLPSA, SQA, and SUMDUMP are the defaults even if other parameters are specified for SDATA. NOALLPSA, NOSQA, and NOSUMDUMP must be specified to suppress these defaults. **Note:** The PSA option is not required because it is dumped as a default in all SVC dumps.
,STORAGE=*(strt addr,end addr)* ,LIST=*list addr* ,LISTA=*list addr*	*strt addr:* RX-type address, or register (2) - (12). *end addr:* RX-type address, registers (2) - (12). *list addr:* RX-type address, or register (2) - (12). **Default:** One or more pairs of addresses may be specified, separated by commas. For example: ,STORAGE=*(strt addr,end addr,strt addr,end addr)*
,SUBPLST=*subpool id list addr*	*subpool id list addr:* RX-type address or register (2) - (12).
,KEYLIST=*storage key list addr*	*storage key list addr:* RX-type address, or register (2) - (12). **Note:** KEYLIST cannot be specified without SUBPLST.
,BUFFER=NO ,BUFFER=YES	
,QUIESCE=YES ,QUIESCE=NO	
,BRANCH=NO ,BRANCH=YES	**Note:** If BRANCH=YES is specified, ASID or ASIDLST must also be specified.
,SUSPEND=NO ,SUSPEND=YES	**Default:** SUSPEND=NO
,SUMLIST=*list addr*	*list addr:* RX-type address, or register (2) - (12).
,SUMLSTA=*list addr*	*list 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 SDUMP macro instruction, with the following exception:

,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 change SDATA areas, 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 gives control program functions the ability to define their recovery in the FRR (functional recovery routine) LIFO stack, which is used during processing of the system recovery manager. Any 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 used to keep information that might be useful to the FRR. The exit and retry routines execute in the same addressing mode as the SETFRR macro expansion and service routine (for MVS/XA). This is the addressing mode of the issuer of the macro instruction.

The issuer of the SETFRR macro instruction can be in any cross memory mode but must be in supervisor state key zero.

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) before using the SETFRR macro instruction.

The MVS/XA support of the SETFRR macro instruction sets the high-order bit of the recovery exit routine address to the addressing mode of the issuer. This bit setting determines the addressing mode of both the recovery exit routine and the retry routine.

The interface to an FRR is described in Volume 1 under "System Environment." Guidelines for writing an FRR appear in Volume 1 under "Recovery Routine Guidelines."

Note: FRRs need not restore registers upon return.

The SETFRR macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede SETFRR.
SETFRR	
b	One or more blanks must follow SETFRR.
A,FRRAD=*FRR addr* R,FRRAD=*FRR addr* D P	*FRR addr:* A-type address, or register (2) - (12).
,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 (2) - (12). **Note:** This parameter may only be specified with A or R above.
,EUT=YES	
,MODE= (FULLXM PRIMARY HOME , GLOBAL LOCAL GLOBAL,LOCAL LOCAL,GLOBAL)	**Default:** MODE=HOME
,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.

,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. If a register is specified, the address of the 24-byte parameter area is placed in the register. This parameter area is associated with the FRR address that 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.

,EUT = YES

used only with A and R, specifies that the new FRR can be used in any environment. EUT = YES is used by routines that are not certain of their environment; for example, a routine that can be called by an SRB or by a task that is executing enabled and might not hold any locks. While the FRR remains in effect, no SVCs can be issued and no new asynchronous exits are dispatched.

,MODE = *options*

specifies the environment in which the FRR is to get control and also, optionally, identifies the FRRs that free critical resources. The normal or expected addressing environment is identified by FULLXM, PRIMARY, or HOME. The restricted or critical resource freeing addressing environment is identified by LOCAL, GLOBAL, or both. Parentheses are not needed if only one option is chosen.

FULLXM

specifies that the FRR exit must be entered in the same cross memory environment that existed when the SETFRR was issued.

PRIMARY

specifies that the FRR exit must be entered in primary addressing mode with both the PASID and SASID the same as the PASID that existed when the SETFRR was issued, the home address space must be unchanged, and the PSW key mask must be the same as when the SETFRR was issued.

HOME

specifies that the FRR exit must be entered in primary addressing mode with PASID = SASID = HASID, and the PSW key mask either the same as that at the time of the error for SRB mode, or TCBPKF for TCB mode.

If neither FULLXM, PRIMARY, nor HOME is coded, HOME is the default.

GLOBAL

specifies that the FRR frees a critical global resource. If the FRR cannot be entered in its normal addressing environment (for example, if the secondary address space is no longer valid), it must be entered in GLOBAL restricted addressing environment to free critical resources. To enter the FRR, the system mode must be one of the following:

- A global spin lock is held.
- The processor is disabled and the current FRR stack is a super stack.
- The processor is disabled and a PSA super bit is on.

If it cannot be entered either as an FRR or as a resource manager, the FRR is skipped.

LOCAL
specifies that the FRR frees a critical local resource. If the FRR cannot be entered in its normal addressing environment then it must be entered in LOCAL restricted addressing environment to free resources.

In order for the FRR to be entered in LOCAL restricted addressing environment, a local lock must be held.

If it cannot be entered either as an FRR or as a resource manager, the FRR is skipped.

,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.

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 (R5) is placed on the FRR stack in the next available FRR entry. On return, register (R2) contains 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=(R5),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

If RELEASE, TYPE=ALL or RELEASE, TYPE=(*reg*) is coded, this macro can be assembled compatibly between MVS/XA and MVS/370 through the use of the SPLEVEL macro instruction. Default processing will result in an expansion of the macro that operates only with MVS/XA. The expansion of SETLOCK without RELEASE, TYPE=ALL or RELEASE, TYPE=(*reg*) will operate on both MVS/XA and MVS/370. See the topic "Selecting the Macro Level" for additional information.

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 PSW key zero. The SETLOCK macro instruction can be used by programs executing in cross memory mode. A DSECT for the PSA (via the IHAPSA mapping macro) must be included in the CSECT using the SETLOCK macro instruction.

SETLOCK can be used to:

- Obtain a specified lock
- Release a specified lock
- Test a specified lock or to determine if the lock is held on the requestor's processor
- Test whether any user holds a lock higher in the locking hierarchy than a user-specified lock

There are two classes of locks: global and local. In addition to the CPU lock, there are two types of locks: spin and suspend. The CPU lock is not a spin lock or a suspend lock; it is a pseudo spin lock. The descriptions of these locks and the hierarchy structure in which these locks are arranged are described under "Locking" in Volume 1. CML (cross memory local) lock means the local lock of an address space other than the home address space. LOCAL lock means the local lock of the home address space. When written in lower case, local lock means any local-level lock, either the LOCAL or a CML lock.

Notes:

1. *In MVS/XA a locked routine is not allowed to issue an SVC, or invoke a routine that would issue an SVC on the locked routine's behalf.*
2. *Caution should be used if SETLOCK is invoked and register 11, 12, 13 or 14 is used as the program's base register. Some options of the SETLOCK macro cause branch instructions to be generated after setting registers 11-14 to the required values.*
3. *MVS/XA does not support either the IOSCAT or the IOSLCH lock. The assembly will fail if a user requests the IOSCAT lock or the IOSLCH lock.*

OBTAIN Option

The OBTAIN option of SETLOCK macro instruction is written as follows:

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

OBTAIN	
,TYPE=RSMGL,ADDR=(11) ,TYPE=VSMFIX ,TYPE=ASM,ADDR=(11) ,TYPE=ASMGL,ADDR=(11) ,TYPE=RSMST,ADDR=(11) ,TYPE=RSMCM,ADDR=(11) ,TYPE=RSMXM,ADDR=(11) ,TYPE=RSMAD,ADDR=(11) ,TYPE=RSM,SCOPE=SHR ,TYPE=RSM,SCOPE=EXCL ,TYPE=VSMPAG ,TYPE=DISP ,TYPE=SALLOC ,TYPE=IOSYNCH,ADDR=(11) ,TYPE=IOSUCB,ADDR=(11) ,TYPE=SRM ,TYPE=TRACE,SCOPE=SHR ,TYPE=TRACE,SCOPE=EXCL ,TYPE=CPU ,TYPE=CMS ,TYPE=CMSEQDQ ,TYPE=CMSSMF ,TYPE=CMSALL ,TYPE=CML,ASCB=(11) ,TYPE=LOCAL	**Note:** SCOPE is valid only with TYPE=RSM or TYPE=TRACE.
,MODE=COND ,MODE=UNCOND	**Note:** MODE cannot be specified with TYPE=CPU.
,MODE=UNCOND,DISABLED	DISABLED must not be specified with TYPE=CPU, TYPE=CMS, TYPE=CMSEQDQ, TYPE=CMSSMF, TYPE=CMSALL, TYPE=CML, or TYPE=LOCAL.
,REGS=SAVE ,REGS=USE	**Note:** Registers 11-14 will be destroyed if this parameter is omitted.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

OBTAIN
specifies that the designated lock is to be obtained on the caller's behalf.

,TYPE=RSMGL ,ADDR=(11)
,TYPE=VSMFIX
,TYPE=ASM ,ADDR=(11)
,TYPE=ASMGL ,ADDR=(11)
,TYPE=RSMST ,ADDR=(11)
,TYPE=RSMCM ,ADDR=(11)
,TYPE=RSMXM ,ADDR=(11)
,TYPE=RSMAD ,ADDR=(11)
,TYPE=RSM ,SCOPE=SHR
,TYPE=RSM ,SCOPE=EXCL
,TYPE=VSMPAG
,TYPE=DISP
,TYPE=SALLOC
,TYPE=IOSYNCH ,ADDR=(11)
,TYPE=IOSUCB ,ADDR=(11)
,TYPE=SRM
,TYPE=TRACE ,SCOPE=SHR
,TYPE=TRACE ,SCOPE=EXCL
,TYPE=CPU
,TYPE=CMS
,TYPE=CMSEQDQ
,TYPE=CMSSMF
,TYPE=CMSALL
,TYPE=CML, ASCB=(11)
,TYPE=LOCAL

specifies the type of lock that is to be obtained on the caller's behalf.

The types available are:

RSMGL is the real storage management global lock. It is a global spin lock used to serialize RSM global control blocks.

VSMFIX is the virtual storage management fixed subpool lock. It is a global spin lock used to serialize fixed subpool storage.

ASM is the auxiliary storage management lock. It is a global spin lock used to serialize use of the ASM control blocks on an address space level.

ASMGL is the auxiliary storage management global lock. It is a global spin lock used to serialize ASM resources on a global level.

RSMST is the real storage management steal lock. It is used to serialize the stealing of unchanged pages.

RSMCM is the real storage management common lock. It is used to serialize RSM resources in the common area. This includes page frames and PCB queues.

RSMXM is the real storage management cross memory lock. It is a global spin lock used to serialize RSM resources on an address space level during cross memory processing.

RSMAD is the real storage management address space lock. It is a global spin lock used to serialize resources on an address space level.

RSM is the real storage management lock. It is a global shared/exclusive spin lock used to serialize RSM resources during recovery and the processing of global functions (such as reading and writing RSM control blocks).

VSMPAG is the virtual storage management pageable subpools lock. It is a global spin lock used to serialize the use of pageable common subpools.

DISP is the global dispatcher lock. It is a global spin lock used to serialize dispatcher functions (such as updating the address space vector table and changing the address space control block dispatching queue) on a global level.

SALLOC is the space allocation lock. It is a global spin lock used to serialize external receiving routines that enable a processor for either an emergency signal or a malfunction alert.

IOSYNCH is the IOS synchronization lock. It is a global spin lock used to serialize the global IOS functions.

IOSUCB indicates an IOS unit control block lock. These locks (one per UCB) are global spin locks used to serialize access and updates to UCBs.

SRM is the system resource manager lock. It is a global spin lock used to serialize SRM control algorithms and associated data.

TRACE is the trace lock. It is a global shared/exclusive spin lock used to serialize the system trace buffer structure.

CPU is the processor lock. It is a pseudo spin lock providing system recognized disablement (that is logical disablement). There is one CPU lock per processor and no processor can request another processor's lock. The lock is always available. Users not holding a spin lock can obtain the CPU lock to become disabled for I/O and external interruptions.

CMS is the cross memory services lock. It is a global suspend lock used to serialize functions between address spaces where this serialization is not provided by one or more of the global spin locks.

CMSEQDQ is the ENQ/DEQ cross memory services lock. It is a global suspend lock used to serialize ENQ/DEQ functions and the use of ENQ/DEQ control blocks.

CMSSMF is the SMF cross memory services lock. It is a global suspend lock used to serialize SMF functions and the use of SMF control blocks.

CMSALL indicates that all the cross memory services locks (CMS, CMSEQDQ, and CMSSMF) are to be obtained.

CML is the cross memory local lock. It is a local level suspend type lock used to serialize resources in an address space other than the home address space.

The requestor of a CML lock must have authority to access the specified address space before requesting the lock. To establish authority, the requestor sets the primary or secondary address space to the one specified by the ASCB=(11) parameter. This address space must be non-swappable before the SETLOCK request.

LOCAL is the lock that serializes resources in the home address space pointed to by PSAAOLD. It is a local level suspend lock.

ADDR=(11)
specifies that the address of the lockword has been loaded into register 11 before the SETLOCK request. This parameter is required for multiple spin type locks and cannot be specified for single locks.

,SCOPE=SHR
,SCOPE=EXCL
indicates the manner in which the lock is held. If SHR is specified, the lock can be held by more than one processor at a time. If EXCL is specified, only one processor can hold the lock.

ASCB=(11)

specifies that the address of the ASCB whose local lock is requested has been loaded into register 11 before the SETLOCK request. This parameter must be specified if TYPE=CML is specified and is valid only for CML lock requests

Note: If the requestor specifies OBTAIN, TYPE=CML and the ASCB=(11) parameter points to the home address space, the request is treated as though the LOCAL lock were being obtained.

The return registers are:

11	Unchanged if ADDR or ASCB is specified, otherwise unpredictable.
12	Unpredictable.
13	Return code.
14	Return address.

,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 on another processor, it is acquired on the caller's behalf. If the lock is already held, control is returned indicating that the caller holds the lock or that another unit of work on another processor owns the lock.

UNCOND specifies that the lock is to be unconditionally obtained. That is, if the lock is not owned on another processor, it is acquired on the caller's behalf. If the lock is already held by the caller, control is returned to the caller indicating that he already owns the lock. If the lock is held on another processor, the caller's processor spins on the lock until it is released or suspends the SETLOCK caller until the lock is available.

DISABLED specifies that the caller is already disabled for I/O and external interruptions.

,REGS=SAVE
,REGS=USE

specifies the use of registers 11 through 14.

SAVE specifies that register contents are to be saved. Registers 11 through 14 are saved in the area pointed to by register 13, and are restored upon completion of the SETLOCK request. The save area consists of at least 5 words (These words hold the contents of the four registers and the return code that is to be placed in register 15).

Note: The save area used for the REGS=SAVE parameter must be a different area than the standard linkage save area used by the program.

USE specifies that registers 14, 15, 0, and 1 are available for use. Registers 11, 12, and 13 are saved in registers 15, 0, and 1, respectively, and are restored upon completion of the SETLOCK request. Register 14 is used as a link register; register 15 contains 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.

When control is returned, register 15 (register 13, if neither SAVE nor USE is specified) contains one of the following return codes:

Hexadecimal Code	Meaning
00	The lock was successfully obtained.
04	The lock was already held by the caller. The lockword id matches the caller's id.
08	The conditional obtain process was unsuccessful. The lockword id does not match the caller's id. This means that the lock is owned by another processor. In the case of a shared/exclusive lock, this return code means that the lock was not immediately available with the scope requested.
10	A level error was detected. This return code is supplied on a conditional obtain only. A level error detected on an unconditional obtain results in an abnormal termination.

Notes:

1. *See the topic "Locking" in Volume 1 for a description of the types of level errors that the lock manager can and cannot detect.*

2. *For an unconditional request, if the caller holds the lockword on a different level, the lock manager abnormally terminates the caller with a 073 ABEND.*

3. *If the RSM lock or the TRACE lock is held shared and requested exclusive on the same processor, or held exclusive and requested shared on the same processor, the lock manager issues a 073 ABEND with reason code of X'24'.*

Example 1

Operation: The global dispatcher lock, DISP, is to be conditionally requested. The RELATED parameter indicates that the DISP lock serializes the ASCB resource, and the lock is either freed at the location represented by NAME or SYM1 in module MODABE or by SYMZ in module MODABC.

```
SETLOCK OBTAIN,TYPE=DISP,MODE=COND,RELATED=(ASCB,          X
        MODABE(NAME,SYM1),MODABC(SYMZ))
```

Release Option

The RELEASE option of the SETLOCK macro instruction is written as follows:

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

RELEASE	
,TYPE=RSMGL,ADDR=(11)	*reg:* decimal digit 2 - 10.
,TYPE=VSMFIX	**Note:** SCOPE is valid only for TYPE=RSM or TYPE=TRACE.
,TYPE=ASM,ADDR=(11)	
,TYPE=ASMGL,ADDR=(11)	
,TYPE=RSMST,ADDR=(11)	
,TYPE=RSMCM,ADDR=(11)	
,TYPE=RSMXM,ADDR=(11)	
,TYPE=RSMAD,ADDR=(11)	
,TYPE=RSM,SCOPE=SHR	
,TYPE=RSM,SCOPE=EXCL	
,TYPE=VSMPAG	
,TYPE=DISP	
,TYPE=SALLOC	
,TYPE=IOSYNCH,ADDR=(11)	
,TYPE=IOSUCB,ADDR=(11)	
,TYPE=SRM	
,TYPE=TRACE,SCOPE=SHR	
,TYPE=TRACE,SCOPE=EXCL	
,TYPE=CPU	
,TYPE=CMS	
,TYPE=CMSEQDQ	
,TYPE=CMSSMF	
,TYPE=CMSALL	
,TYPE=LOCAL	
,TYPE=SPIN	
,TYPE=ALL	
,TYPE=*(reg)*	
,TYPE=CML,ASCB=(11)	
,DISABLED	**Note:** DISABLED cannot be specified with TYPE=CMS, TYPE=CMSEQDQ, TYPE=CMSSMF, TYPE=CMSALL, TYPE=CML, TYPE=LOCAL, or TYPE=CPU
,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 lock is to be released.

,TYPE=SPIN
,TYPE=ALL
,TYPE=*(reg)*

specifies the type of lock that is to be released.

SPIN indicates that all spin locks currently held on the processor are to be released.

ALL indicates that all locks currently held on the processor are to be released.

(reg) indicates that 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	Lock
00	CPU
01	Reserved
02	Reserved
03	Reserved
04	RSM
05	TRACE
06	Reserved
07	Reserved
08	Reserved
09	Reserved
10	Reserved
11	RSMCM
12	RSMGL
13	VSMFIX
14	ASMGL
15	RSMST
16	RSMXM
17	RSMAD
18	VSMPAG
19	DISP
20	ASM
21	SALLOC
22	IOSYNCH
23	Reserved (previously IOSCAT)
24	IOSUCB
25	Reserved (previously IOSLCH)
26	Reserved
27	Reserved
28	Reserved
29	SRM
30	Any cross memory services lock held
31	LOCAL or CML lock

Notes:

1. *It is highly recommended that users who specify TYPE=ALL or TYPE=(reg) recompile. Such users must recompile if they hold a lock that is new to MVS/XA at the time of the SETLOCK RELEASE request.*

2. *You must specify TYPE=CMSALL to obtain all the CMS locks. To release all CMS locks, you must specify RELEASE, TYPE=CMSALL. You cannot specify RELEASE, TYPE=CMS if you own all of the CMS locks.*

3. *If you specify RELEASE, TYPE=CML and the ASCB=(11) parameter specifies the home address space and the lock you are holding is home's local lock, then SETLOCK processing treats the CML release request as a RELEASE, TYPE=LOCAL.*

4. *If the CPU lockword is held on a processor, RELEASE,TYPE=ALL and RELEASE,TYPE=SPIN cause the lockword to be set to zero; RELEASE,TYPE=(reg) causes the lockword to be decreased by one.*

5. *The SCOPE keyword cannot be specified with the following options:*

   RELEASE, TYPE=ALL
   RELEASE, TYPE=SPIN
   RELEASE, TYPE=(reg)

   If a shared/exclusive lock is to be released via one of these invocations, the shared/exclusive lock will be released with the scope currently in effect on the processor.

,DISABLED

specifies that control is to be returned to the caller with the processor physically disabled for I/O and external interruptions 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 have recursion constraints. When control is returned, register 15 (register 13 if neither SAVE nor USE is specified) 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. This means that the lock was owned by a different processor.
0C	The release process was unsuccessful. The caller does not own the specified local or CML lock. This return code applies to LOCAL or CML release only.

The following return codes refer to multiple locks:

- If TYPE=CMSALL is specified, the return code will be as described above as long as the caller owns either both of the cross memory services locks, or none at all. Users, who own only one of the cross memory services locks, but specify CMSALL on the release, will be abended.

- If TYPE=ALL or TYPE=(reg) is specified, the return code is X'00'.

- If TYPE=SPIN is specified, the return code is X'00' for success or X'04' if no spin locks were held at entry to the service routine.

Example 1

Operation: Release the local lock.

```
SETLOCK RELEASE,TYPE=LOCAL,RELATED=(TCBRQ,MOD1(NAME1),    X
  MOD2(NAME2))
```

Example 2

Operation: Release the IOSUCB lock whose address is in register 11.

```
SETLOCK RELEASE,TYPE=IOSUCB,ADDR=(11),RELATED=(AXYZ,MOD1(LABEL))
```

TEST Option

The TEST option of the SETLOCK macro instruction is written as follows:

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

TEST	
,TYPE=RSMGL ,TYPE=VSMFIX ,TYPE=ASM ,TYPE=ASMGL ,TYPE=RSMST ,TYPE=RSMCM ,TYPE=RSMXM ,TYPE=RSMAD ,TYPE=RSM,SCOPE=SHR ,TYPE=RSM,SCOPE=EXCL ,TYPE=VSMPAG ,TYPE=DISP ,TYPE=SALLOC ,TYPE=IOSYNCH ,TYPE=IOSUCB ,TYPE=SRM ,TYPE=TRACE,SCOPE=SHR ,TYPE=TRACE,SCOPE=EXCL ,TYPE=CPU ,TYPE=CMS ,TYPE=LOCAL ,TYPE=SPIN ,TYPE=ALL ,TYPE=*(reg)* ,TYPE=CML ,TYPE=ALOCAL ,TYPE=HIER	*reg:* any valid register value
,ADDR=*(reg)*	**Note:** ADDR=*(reg)* can only be specified with: TYPE=RSMGL, TYPE=RSMXM TYPE=ASM, TYPE=RSMAD TYPE=ASMGL, TYPE=IOSYNCH TYPE=RSMST, TYPE=IOSCUCB TYPE=RSMCM
,ASCB=*(11)*	**Note:** ASCB can only be specified with TYPE=CML.
,LOCKHLD=*(reg)*	**Note:** LOCKHLD can only be specified with TYPE=CML, TYPE=ALOCAL, TYPE=CPU
,LOCK=*lockname*	**Note:** LOCK=*lockname* is required and can only be specified with TYPE=HIER. *lockname:* any lock except: LOCAL, ALOCAL, CMSSMF CML, CPU CMS, CMSEQDQ
,BRANCH=(HELD,*addr*) ,BRANCH=(NOTHELD,*addr*)	*addr:* RX-address. *addr* cannot be specified with TYPE=HIER.
,BRANCH=(HELD,*label*) ,BRANCH=(NOTHELD,*label*)	*label:* symbol. *label* can only be specified with TYPE=HIER.
,REGS=*(reg)* ,REGS=*(reg1,reg2)*	*reg:* decimal digit 2-12. **Note:** *(reg)* can only be specified with TYPE=SPIN, TYPE=ALL, or TYPE=*(reg)* *reg1:* any valid register value except 0. *reg2:* any valid register value. **Default:** If *(reg1,reg2)* is not specified, the default is (11,12). **Note:** *(reg1,reg2)* can only be specified with TYPE=HIER.
,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 designated lock is to be checked to determine if it is currently held on the requesting processor.

,TYPE=SPIN
specifies that the caller wants to determine whether he holds any spin locks or the CPU lock.

,TYPE=ALL
specifies that the caller wants to determine whether he holds any locks.

,TYPE=(*reg*)
specifies that the caller wants to determine whether he owns the locks indicated by the bit setting in the specified register.

,TYPE=CML
specifies that the requestor wishes to determine whether a CML lock is held. The ASCB=(11) parameter or the LOCKHLD=*(reg)* parameter must be specified with TYPE=CML.

,TYPE=ALOCAL
specifies that the requestor wishes to determine whether a local lock is held, either home's LOCAL or a CML. The LOCKHLD=*(reg)* parameter may be specified with TYPE=ALOCAL. ASCB may not be specified with TYPE=ALOCAL.

,TYPE=HIER
indicates that a check is to be made to determine whether the requesting processor owns any locks higher in the locking hierarchy than that specified by the parameter, LOCK=*lockname*.

,ADDR=*(reg)*
specifies that the designated register contains the lockword address to be used to determine if the specified lock is owned by the caller. This parameter is only valid for multiple spin type locks but is not a required parameter.

,ASCB=(11)
specifies that the register 11 contains the ASCB address that is to be checked to determine if the requestor owns its local lock as a CML lock. This parameter is only valid with TYPE=CML.

,LOCKHLD=*(reg)*
specifies that the a designated register is to be used as a work register by the macro. This parameter is valid only for TYPE=CML, TYPE=ALOCAL, and TYPE=CPU.

If TYPE=CML is specified, this register is loaded with the contents of PSALOCAL. If a CML lock is held, this register will contain the ASCB address of the CML locked address space.

If TYPE=ALOCAL is specified, this register is loaded with the contents of PSALOCAL. If the LOCAL lock is held, this register will contain zero.

If TYPE=CPU is specified, this register will be loaded with the current CPU lock use count for this processor.

,LOCK=*lockname*
is used with TYPE=HIER to determine whether the processor holds any locks higher than *lockname*.

,BRANCH=(HELD,*addr***)**
,BRANCH=(NOTHELD,*addr***)**
,BRANCH=(HELD,*label***)**
,BRANCH=(NOTHELD,*label***)**
specifies that the return code setting of the macro instruction is to be suppressed and replaced by a direct branch to the specified address or the specified label.

If (HELD,*addr*) is specified, the address is 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 on the requesting processor.

If (NOTHELD,*addr*) is specified, the address is branched to if the specified lock is not currently held on the requesting processor, or if not all the locks specified for TYPE=(*reg*) are held, or if no lock for TYPE=ALL or TYPE=SPIN is held.

(HELD,*label*) indicates that program execution will continue at the label specified if any higher locks are held. This can be specified only with TYPE=HIER.

(NOTHELD,*label*) indicates that program execution will continue at the label specified if no locks higher than the lock specified are held. This can be specified only with TYPE=HIER.

,REGS=(*reg***)**
,REGS=(*reg1,reg2***)**
(*reg*) is used only with TYPE=SPIN, TYPE=ALL, and TYPE=(*reg*). After the macro executes, the register specified contains 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 input string is combined with the PSACLHS field by an AND instruction and the results are returned in the register specified.

(*reg1,reg2*) is used with TYPE=HIER. The registers specified are two unique general purpose registers to be used as work registers in the code generated by the SETLOCK TEST expansion. If REGS is not specified, the default is (11,12).

When control is returned, register 15 contains one of the following return codes (if the BRANCH= parameter was omitted):

Hexadecimal Code	Meaning
00	The lock was held by the requestor, all the locks were held (if the request was for several locks via a register), at least one lock was held (if TYPE=CMS, TYPE=ALL, or TYPE=SPIN was specified), or no locks higher in the hierarchy than a user-supplied lock were held.
04	The lock was not held by anybody, not all the locks were held (if the request was for several locks via a register), no lock was held (if TYPE=CMS, TYPE=ALL, or TYPE=SPIN was specified), or at least one lock higher in the hierarchy than a user-specified lock was held.

Notes:

1. *TYPE=CMS is used to determine if at least one cross memory services lock is held, but cannot be used to determine which one or if all are held.*

2. *If you specify the ADDR=(reg) parameter for a multiple spin type lock, then the designated register contents are compared to the appropriate current lock held table slot (PSACLHT).*

3. *If you do not specify the ADDR=(reg) parameter, then only the appropriate PSACLHS bit is used to determine whether the lock is currently owned by the requestor.*

4. *If you specify TYPE=ALOCAL without the ASCB or LOCKHLD parameter, then only the PSACLHS bit for the local lock is used to determine whether you currently own any local lock. The LOCKHLD parameter permits you to extract the CML lock indicator from the PSALOCAL field and load it into a designated register. PSALOCAL, in conjunction with the local lock bit in PSACLHS, indicates whether you hold any local lock as well as whether it is a CML lock or the LOCAL lock.*

Example 1

Operation: If a local lock is not held, branch to DSRLLINT, otherwise, execute the next sequential instruction.

```
SETLOCK TEST,TYPE=LOCAL,BRANCH=(NOTHELD,DSRLLINT)
```

Example 2

Operation: Put the current CPU lock use count for this processor into register 3.

```
SETLOCK TEST,TYPE=CPU,LOCKHLD=(3)
```

Example 3

Operation: Branch to the label HERE if the processor does not own any locks higher in the locking hierarchy than the RSM lock; otherwise execute the next sequential instruction.

```
SETLOCK TEST,TYPE=HIER,LOCK=RSM,BRANCH=(NOTHELD,HERE)
```

SETRP - Set Return Parameters

If you are executing in 31-bit addressing mode, you must use the MVS/XA version of this macro instruction.

The SETRP macro instruction is used to indicate the various requests that a recovery routine can make. It may be used only if a system diagnostic work area (SDWA) was passed to the recovery routine. The macro instruction is valid only for FRRS and ESTAE type recovery routines.

The description of the SETRP macro instruction follows. The SETRP macro instruction is also described in *Supervisor Services and Macro Instructions* with the exception of the RECORD, RECPARM, FRELOCK, CPU, SERIAL, and RETRY parameters. These parameters are restricted in use to programs executing as functional recovery routines in supervisor state or key 0-7 and, therefore, are only described here.

Note: This macro instruction requires that the IHASDWA mapping macro be assembled as a DSECT in the caller's program. The SDWA is addressable when the recovery routine is entered; when the SETRP macro instruction is issued, the same address space must be addressable.

The SETRP macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede SETRP.
SETRP	
b	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. If RETREGS=YES is specified for a FRR, all registers are restored from SDWASRSV with the exception of register 15. Register 15 always contains the entry point of the retry routine.
,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: RMS(*lockword*) SRM VSMFIX TRACE ASMGL(*lockword*) CPU RSMST(*lockword*) SALLOC RSMCM(*lockword*) CMS RSMXM(*lockword*) LOCAL RSMAD(*lockword*) IOSUCB(*lockword*) RSM IOSYNCH(*lockword*) VSMPAG ASM(*lockword*) DISP CML(*cmlasch*) *cmlascb:* RX-type address or register (2) - (12). *lockword:* RX-type address. **Note:** This parameter may be specified only if RC=0 is specified above.

,REASON=*code*	*code:* symbol, decimal or hexadecimal number, or register (2) - (12).
,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.
,SERIAL=YES ,SERIAL=NO	
,RETRY=FRR ,RETRY=ERROR	**Default:** RETRY=FRR
,RETRY15=NO ,RETRY15=YES	**Default:** RETRY15=NO
,REMREC=NO ,REMREC=YES	**Default:** REMREC=NO
,FRLKRTY=NO ,FRLKRTY=YES	**Default:** FRLKRTY=NO

The parameters are explained below:

,WKAREA=(*reg*)
specifies the address of the SDWA passed to the recovery exit. If this parameter is omitted, the address of the SDWA must be in register 1.

,REGS=(*reg* 1)
,REGS=(*reg* 2)
specifies the register or range of registers to be restored from the save area pointed to by the address in register 13. If REGS is specified, a branch on register 14 instruction will also be generated to return control to the control program. If REGS is not specified, the user must code his own return.

,DUMP=IGNORE
,DUMP=YES
,DUMP=NO
specifies that the dump option fields will not be changed (IGNORE), will be zeroed (NO), or will be merged with dump options specified in previous dump requests, if any (YES). If IGNORE is specified, a previous exit had requested a dump or a dump had been requested via the ABEND macro instruction, and the previous request will remain intact. If NO is specified, no dump will be taken.

,DUMPOPT=*parm list addr*
specifies the address of a parameter list that is valid for the SNAP macro instruction. The parameter list can be created by using the list form of the SNAP macro instruction, or a compatible list can be created. If the specified dump options include subpools for storage areas to be dumped, up to seven subpools can be dumped. Subpool areas are accumulated and wrapped, so that the eighth subpool area specified replaces the first. The TCB, DCB, and STRHDR options available on SNAP are ignored if they appear in the parameter list. The TCB used will be the one for the task that suffered the error. The DCB used will be one created by the control program, and either SYSABEND, SYSMDUMP, or SYSUDUMP will be used as a DDNAME.

,REASON=*code*

specifies the reason code that the user wishes to pass to subsequent recovery exits. The value range for *code* is any 32-bit hexadecimal number or 31-bit decimal number. See *Supervisor Services and Macro Instructions* for information about how a user can change this code.

,RC=0
,RC=4
,RC=16

specifies the return code the recovery routine sends to RTM to indicate what further action is required:

```
0  -  Continue with termination, causes entry into
      previously-specified recovery routine, if any.
4  -  Retry using the retry address specified.
16 -  Suppress further ESTAI/STAI processing
      (for ESTAI only)
```

,RETADDR=*retry addr*

specifies the address of the retry routine to which control is to be given.

,RETREGS=NO
,RETREGS=YES
,RETREGS=YES ,RUB=*reg info addr*

specifies the contents of the registers on entry to the retry routine. If NO is specified, explicitly or as a default, only parameter registers (14-2) are passed; all others are unpredictable. If YES is specified, the contents of the SDWASRSV field will be used to initialize registers 0-14 when an FRR requests retry and registers 0-15 when an ESTAE requests retry. For ESTAE exits, this field contains the registers at the last interruption of the RB level at which retry will occur. For ESTAI exits, the contents of SDWASRSV must be set by the user either before SETRP is issued or by use of the RUB parameter; any field not set will cause the corresponding register to contain 0 on entry to the retry routine.

RUB specifies the address of an area (register update block) that contains register update information. RTM will move the data specified in this area into the SDWA and into the general purpose registers before entry to the retry routine.

The maximum length of the RUB is 66 bytes:

- The first two bytes represent the registers to be updated, register 0 corresponding to bit 0, register 1 corresponding to bit 1, and so on. The user indicates which of the registers are to be stored in the SDWA by setting the corresponding bits in these two bytes.

- The remaining 64 bytes contain the update information for the registers, in the order 0-15. If all 16 registers are being updated, this field consists of 64 bytes. If only one register is being updated, this field consists of only 4 bytes for that one register.

For example, if only registers 4, 6, and 9 are being updated:

- Bits 4, 6, and 9 of the first two bytes are set.

- The remaining field consists of 12 bytes for registers 4, 6, and 9; the first 4 bytes are for register 4, followed by 4 bytes for register 6, and 4 final bytes for register 9.

,FRESDWA=NO
,FRESDWA=YES
specifies that the entire SDWA be freed (YES) or not be freed (NO) before entry into the retry routine.

,COMPCOD=*comp code*
,COMPCOD=(*comp code***,USER)**
,COMPCOD=(*comp code***,SYSTEM)**
specifies the user or system completion code that the user wants to pass to subsequent recovery exits.

,FRELOCK=*(locks)*
specifies the locks to be freed and the corresponding lockwords that are placed in the SDWA:

RSMGL(*lockword*)	-	Real storage management global lock
VSMFIX	-	Virtual storage management fixed subpool lock
ASMGL(*lockword*)	-	Auxiliary storage management global lock
RSMST(*lockword*)	-	Real storage management steal lock
RSMCM(*lockword*)	-	Real storage management common lock
RSMXM(*lockword*)	-	Real storage management cross memory lock
RSMAD(*lockword*)	-	Real storage management address space lock
RSM	-	Real storage management lock
VSMPAG	-	Virtual storage management subpool lock
DISP	-	Global dispatcher lock
SRM	-	Systems resource manager lock
TRACE	-	Trace lock
CPU	-	Processor lock
SALLOC	-	Space allocation lock
CMS	-	Cross memory services lock
LOCAL	-	Storage lock of the storage the caller is executing in
IOSUCB(*lockword*)	-	IOS unit control block lock
IOSYNCH(*lockword*)	-	IOS synchronization lock
ASM(*lockword*)	-	Auxiliary storage management lock
CML(*cmlascb*)	-	Cross memory local lock, where *cmlascb* indicates the ASCB address of the address space for which the local lock is to be freed

Note: If FRLKRTY=NO is specified or taken as a default, the specified locks are freed only on percolation, not on retry. Specifying FRLKRTY=YES allows the locks listed in FRELOCK to be freed on retry.

,CPU=(*reg***)**
specifies the register that contains the logical processor identification of the processor holding the resource that this processor is waiting for.

,RECORD=IGNORE
,RECORD=YES
,RECORD=NO
specifies that the entire SDWA (fixed, base, variable areas, and extensions) 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 used to update the SDWA with recording information. The parameter list consists of three 8-byte fields:

- The first field contains the load module name.

- The second field contains the CSECT name (assembly module name).
- The third field contains the recovery routine name (assembly module name). If the recovery routine label is not the same as the assembly module name, the label can be placed in the SDWARRL field.

The three fields are left-justified, and padded with blanks.

,SERIAL=YES
,SERIAL=NO
specifies whether the percolation from an SRB mode FRR to a related task recovery routine (ESTAE or FRR) is to be serialized (YES) or not serialized (NO) with respect to unlocked task recovery. See "SRB to Task Percolation" in Volume 1.

If the task is already in recovery for another error when SERIAL=YES is specified, the percolation request is deferred pending a requested task retry from any recovery routine covering mainline code. If such a retry is not requested, the task is terminated and all deferred percolations are purged. Only the last FRR to receive control when an error occurs can specify SERIAL=YES.

,RETRY=FRR
,RETRY=ERROR
specifies the cross memory environment in which the retry routine gets control.

RETRY=FRR, the default, specifies that the retry routine gets control in the cross memory environment that exists at the time of entry to the FRR.

RETRY=ERROR specifies that the retry routine gets control in the cross memory environment that existed at the time of the error. Do not specify RETRY=ERROR if the cross memory status at the time of the error is not available, that is, if SDWARPIV is set to one. (Be careful not to create a loop by retrying to an erroneous cross memory state with RETRY=ERROR.)

,RETRY15=YES
,RETRY15=NO
In an FRR environment only, specifies that register 15 is restored from SDWASRSV if RETRY15=YES. Otherwise, it contains the entry point address of the retry routine.

This parameter may be specified only when RC=4 is specified. If RETRY15=YES is not coded on any SETRP invocation prior to returning to RTM, the effect is that of specifying RETRY15=NO.

,REMREC=YES
,REMREC=NO
In an FRR or ESTAE environment, specifies that the FRR/ESTAE entry for the currently running FRR/ESTAE routine be removed (REMREC=YES) or not removed (REMREC=NO). This parameter may be specified only when RC=4 is specified, indicating a retry request.

The entry is removed before control returns to the retry point. If REMREC=YES is not coded on any SETRP invocation before RTM receives control, the effect is that of specifying REMREC=NO. The REMREC parameter may be used to remove a recovery routine that has been established with a token, although the token cannot be specified when you code the SETRP macro.

,FRLKRTY=YES
,FRLKRTY=NO

In an FRR environment only, specifies that the locks specified on FRELOCK be freed (FRLKRTY=YES) or not be freed (FRLKRTY=NO) on retry.

This parameter may be specified only when RC=4 is specified. If FRLKRTY=YES is not coded on any SETRP invocation prior to returning to RTM, the effect is that of specifying FRLKRTY=NO.

Notes:

1. *The variable recording area (SDWAVRA) contains the variable information that is supplied by the user. This consists of footprints or other information about the execution environment at the time of the failure. The execute form of the VRADATA macro and the IHAVRA mapping macro can be used to supply this data in a key-length-data format to simplify later decoding. The variable recording area is preceded by the following control information:*

   - *A two-byte length field (SDWAVRAL), filled in by the system, specifying the total length available to the user. This is 255 bytes, the length of the SDWAVRA field.*

   - *A one-byte flag field (SDWADPVA), filled in by the user, specifying the format of the data to be dumped. The flags used to specify the format are:*

     SDWAHEX for hexadecimal format
     SDWAEBC for EBCDIC format
     SDWAVRAM for key-length-data

     More than one of these flags can be set. If the SDWAEBC flag is set, the EREP program formats the SDWAVRA in EBCDIC and hexadecimal for SYS1.LOGREC output.

   - *A one-byte length field (SDWAURAL), filled in by the user, specifying the actual length of the data.*

2. *The FRESDWA parameter cannot be specified or defaulted for a functional recovery routine (FRR). The SDWA is always released before an FRR's retry routine gets control.*

3. *The SERIAL parameter is relevant only for FRRs established for SRBs that have a related task.*

4. *The SERIAL and RETRY parameters are mutually exclusive.*

5. *SETRP does the following in response to requests to alter the completion code and/or reason code:*

   - *If the COMPCOD parameter is altered, SETRP places the new completion code in the SDWACMPC field of the SDWA and sets the SDWACCF flag to indicate that a recovery exit altered the completion code.*

   - *If the REASON parameter is specified, SETRP places the new reason code in SDWACRC and sets the SDWAREAF flag to indicate that a recovery exit altered the reason code.*

The following table indicates which parameters are available to functional recovery routines (FRRs) and which parameters are available to ESTAE-type recovery routines.

Parameter	FRR	ESTAE-type recovery routines
WKAREA	x	x
REGS	x	x
DUMP	x	x
REASON	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
SERIAL	x	
RETRY	x	

Example 1

Operation: Cause a restart interruption on the processor identified by the contents of register 7. In this example, the interrupted function is spinning on a lock currently being held by the processor identified in register 7.

```
SETRP CPU=(7)
```

Example 2

Operation: The first FRR established for an SRB routine requests percolation, freeing of the CML lock (the ASCB address is in register 2), and serialization of percolation to the related task.

```
SETRP   RC=0,FRELOCK=(CML(2)),SERIAL=YES
```

Example 3

Operation: An FRR requests retry with the retry routine getting control in the same cross memory mode as the time of FRR entry. The retry address is in register 3.

```
SETRP   RC=4,RETADDR=(3),RETRY=FRR
```

SPIE - Specify Program Interruption Exit

The SPIE macro instruction specifies the address of an interruption exit routine and the program interruption types that are to cause the exit routine to get control. If the program interruption types specified can be masked, the corresponding program mask bit in the PSW (program status word) is set to 1.

Only callers in 24-bit addressing mode can issue the SPIE macro instruction. If a caller in 31-bit addressing mode issues a SPIE macro, the caller is abended with a system completion code of X'30E'. Callers in 31-bit addressing mode must use the ESPIE macro instruction, which performs the same function as the SPIE macro instruction for callers in both 24-bit and 31-bit addressing mode. For additional information concerning the relationship between the SPIE and the ESPIE macro instructions, see the section on program interruption processing in Volume 1.

Note: In MVS/370 the SPIE environment existed for the life of the task. In MVS/XA, the SPIE environment is deleted when the request block that created it is deleted. That is, when a program running under MVS/XA completes, any SPIE environments created by the program are deleted. This might create an incompatibility with MVS/370 for programs that depend on the SPIE environment remaining in effect for the life of the task rather than the request block.

The SPIE macro instruction is not supported in cross memory mode.

The following description of the SPIE macro instruction also appears in *Supervisor Services and Macro Instructions* with the exception of interruption type 17. This interruption type designates page faults and its use is restricted to an installation-authorized system programmer. For more information about the SPIE macro instruction, see the section on "Processing Program Interruptions" in Volume 1.

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

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

exit addr	
,(interrupts)	*exit addr:* A-type address, or register (2) - (12). *interrupts:* decimal numbers 1-15, or 17 expressed as **single values:** (2,3,4,7,8,9,10) **ranges of values:** ((2,4),(7,10)) **combinations:** (2,3,4,(7,10))

The parameters are explained as follows:

exit addr
specifies the address of the exit routine to be given control when a specific program interruption occurs. The exit routine receives control in 24-bit addressing mode.

,(*interrupts*)
indicates the type of interruption for which the exit routine is to be given control. The interruption types are as follows:

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

Notes:

1. *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.*

2. *The control program returns the address of the previous PICA or fake PICA in register 1. If no previous SPIE environment existed, the control program returns zeros in register 1.*

3. *If an exit address is zero or no parameters are specified, the SPIE environment is canceled.*

4. *For both ESPIE and SPIE — If you are using vector instructions and an interruption of 8, 12, 13, 14, or 15 occurs, your recovery routine can check the exception extension code (the first byte of the two-byte interruption code in the EPIE or PIE) to determine whether the exception was a vector or scalar type of exception.*

Example 1

Operation: Give control to an exit routine for interruption 17. DOITSPIE is the address of the SPIE exit routine.

```
SPIE DOITSPIE,(17)
```

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.
b	One or more blanks must precede SPIE.
SPIE	
b	One or more blanks must follow SPIE.
exit addr	*exit addr:* A-type address.
,(interrupts)	*interrupts:* decimal numbers 1-15, or 17, expressed as **single values:** (2,3,4,7,8,9,10) **ranges of values:** ((2,4),(7,10)) **combinations:** (2,3,4,(7,10))
,MF=L	

The parameters are explained under the standard form of the SPIE macro instruction, with the following exception:

,MF=L
specifies the list form of the SPIE macro instruction.

SPIE (Execute Form)

A remote control program parameter list 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.
b	One or more blanks must precede SPIE.
SPIE	
b	One or more blanks must follow SPIE.
exit addr	*exit addr:* RX-type address, or register (2) - (12).
,(interrupts)	*interrupts:* decimal numbers 1-15, or 17, expressed as **single values:** (2,3,4,7,8,9,10) **ranges of values:** ((2,4),(7,10)) **combinations:** (2,3,4,(7,10))
,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 exception:

,MF=(E,*ctrl*,*addr*)
specifies the execute form of the SPIE macro instruction using a remote control program parameter list.

Note: If SPIE is coded with a 0 as the control address, the SPIE environment is canceled.

SPLEVEL - Set and Test Macro Level

Specific macro instructions supplied in the MVS/XA macro library are identified as downward incompatible (to MVS/370). Unless the user takes specific action, these macros generate downward incompatible statements. It is possible to cause the generation of downward compatible expansions of these macro instructions by using the SPLEVEL macro instruction. The downward incompatible macro instructions interrogate a global set symbol (set by SPLEVEL) during assembly to determine the type of expansion to be generated. See the topic "Selecting the Macro Level" for additional information concerning the downward incompatible macro instructions, and *Assembler H Version 2 Application Programming: Language Reference* for information about global set symbols.

The SPLEVEL macro instruction is written as follows:

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

SET=*n* SET TEST	*n*: 1 or 2 **Default:** SET=2

The parameters are explained as follows:

SET=*n*
SET
TEST
specifies whether the macro level is being set or tested.

If SET=*n* is specified, the SPLEVEL routine sets a global set symbol equal to *n*, where *n* must be 1 or 2. If a user codes one of the downward incompatible macros, one of the following macro expansions is generated:

- the MVS/370 (MVS System Products Version 1 Release 3 level) macro expansion if *n*=1
- the MVS/XA macro expansion if *n*=2

If SET is specified without *n*, the SPLEVEL routine uses the default value, which is 2.

The TEST option is used to determine the macro level that is in effect. The results of the test request are returned to the user in the global set symbol, &SYSSPLV. If TEST is specified and if SPLEVEL SET has not been issued during this assembly, the SPLEVEL routine puts the default value into the global set symbol. If SPLEVEL SET has been issued, the previous value of *n* or the default value is already in the global set symbol.

Example 1

Operation: Select the MVS/370 version of a specific downward incompatible macro instruction.

```
SPLEVEL SET=1
```

Example 2

Operation: Select the MVS/XA version of a specific downward incompatible macro instruction.

```
SPLEVEL SET=2
```

SPOST - Synchronize POST

The SPOST macro instruction is used in a cross-memory post environment to ensure that all outstanding cross-memory post requests to the current address space have completed. SPOST resolves a synchronization problem that arises when it becomes necessary to free an ECB that is a potential target for a cross-memory post request. Before issuing SPOST, you must stop any new posts from being initiated.

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.
b	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
```

SRBSTAT - Save, Restore, or Modify SRB Status

The SRBSTAT macro instruction allows the caller to save, restore, and modify the status of an SRB in a caller-supplied save area. The caller must be running in SRB mode to use the SAVE or RESTORE option. The caller can be running either in SRB or TCB mode to use the MODIFY option. The caller must be in supervisor state, key 0, have authority to issue a SSAR to the home address space, and must be enabled and unlocked. Register 13 must point to a 72-byte save area addressable in the primary address space. Control returns from the SRBSTAT macro instruction in primary mode.

The SRBSTAT macro instruction is written as follows:

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

SAVE RESTORE MODIFY	
,STSV=*stsv addr* ,STSV=0	*stsv addr:* RX-type address or register (1) - (12), register (1) preferred.
,NEWFRR=*addr*	*addr:* RX-type address or register (0) or (2) - (12), register (0) preferred.
,PRGAT=*pat addr*	*pat addr:* RX-type address or register (2) - (12), register (2) preferred.

The parameters are explained as follows:

SAVE
RESTORE
MODIFY
specifies whether a save, restore, or modify operation is requested. For SAVE or RESTORE, the following status is saved or restored:

- General and floating point registers
- Control registers 3 and 4
- CPU affinity mask
- Related ASID/TCB
- Timing information
- FRR stack
- PCLINK stack header

If SAVE is specified, only caller's registers 1 and 15 are destroyed. Register 1 is used to hold an FRR parameter area address if NEWFRR is also specified and register 15 is used for a return code. The PCLINK stack header is saved and zeroed.

If RESTORE is specified, registers 0-13 are restored. The contents of register 14 are the same as when RESTORE was entered. The current PCLINK stack header must be zero; the saved one is restored.

On entry to RESTORE, the PCLINK stack header must be zero. RESTORE cannot be used by RTM1 or in an FRR. Note that RESTORE returns to its caller and not to the caller of SAVE.

,STSV = *stsv addr*

specifies the address of the save area to be used for the SAVE, RESTORE, or MODIFY operation. The save area can be in private pageable storage, but it must be addressable from the home address space and it must begin on a double word boundary. The SVT field SVTSSTSV contains the length of the save area. For RESTORE or MODIFY, the save area must contain valid status.

,STSV = 0

specifies that the current status in the LCCA is to be modified. This parameter is valid only with MODIFY.

For MODIFY, an existing SRB status save area or the current status in the LCCA is modified. Only the purge ASID/TCB information can be modified. All registers are saved and restored except register 15, which contains a return code.

Hexadecimal Code	Meaning
00	The modify function was successfully completed.

,NEWFRR = *addr*

specifies the address of an FRR established with MODE = FULLXM. For SAVE, the address of the FRR parameter area is returned to the caller in register 1. The first word of the parameter area contains the address of the SRB status save area being used.

For RESTORE, the FRR address is used only if the saved status cannot be reinstated on the current processor. An SRB with the FRR option is scheduled specifying this FRR.

For MODIFY, this parameter is invalid.

,PRGAT = *pat addr*

specifies the address of a 6-byte area of storage, currently addressable in the primary address space, that contains the new purge ASID/TCB. Bytes 1 and 2 contain the ASID; bytes 3-6 contain the TCB address. This parameter is required with MODIFY but is invalid with SAVE or RESTORE.

SRBTIMER - Establish Time Limit for System Service

The SRBTIMER macro instruction is used to establish a time limit for a system service running in SRB mode. Time accumulates while the service is running; when the time limit expires, the service abends with a system completion code of 05B. The service can retry following the 05B ABEND.

The caller can cancel an established time limit by reissuing the macro instruction and specifying a time limit of zero. The caller can also override the established time limit with a subsequent SRBTIMER macro instruction.

The caller must be in supervisor state, SRB mode, and key 0. Register 13 must point to a 72-byte save area. The SRBTIMER macro instruction can be issued in any addressing mode. The save area must be addressable in the addressing mode in which the macro is issued.

The SRBTIMER macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede SRBTIMER.
SRBTIMER	
b	One or more blanks must follow SRBTIMER.
LIMIT=*stor addr*	*stor addr:* RX-type address or register (0) or (2) - (12).
,ERRET=*err rtn addr*	*err rtn addr:* RX-type address or register (2) - (12).

The parameters are explained as follows:

LIMIT=*store addr*
specifies the virtual storage address of a doubleword field on a doubleword boundary that contains the time limit. The time limit is in the form of a signed 64-bit binary number and must be positive in order for time to elapse. A negative number causes immediate expiration of the time limit. Bit 51 of the binary number is approximately equivalent to one microsecond. If you specify a value greater than 208 days, the control program changes the value to 208 days. The resolution of the timer is model dependent. See *Principles of Operation* for details concerning the timer facility.

,ERRET = *err rtn addr*

specifies the address of the routine to be given control when the SRBTIMER function encounters damaged clocks.

Register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	The time limit was successfully established.
04	The current processor has an operative CPU timer, but not all processors have an operative CPU timer.
08	The current processor has an inoperative CPU timer, but not all processors have an inoperative CPU timer.
0C	All processors in the system have an inoperative CPU timer.
10	The issuer is not in SRB mode. No timing is performed.

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 OS/360 MFT and OS/360 MVT. However, it is recommended that ESTAE be used. The STAE macro instruction is not supported for users executing in 31-bit addressing mode. Such users will be abended.

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

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

0 *exit addr*	*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 as follows:

0
exit addr
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 are not saved when the STAE exit is taken (HALT), that I/O processing is allowed to continue normally when the STAE exit is taken (NONE), or that all outstanding requests for I/O operations are 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 handle the outstanding I/O requests.

Note: If any IBM-supplied access method, except EXCP, is being used, the PURGE=NONE option is recommended. If you use PURGE=NONE, all control blocks affected by input/output processing can 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 develops when an input/output interruption occurs, even if the exit routine is in progress. Thus, it appears that the exit routine failed when, in reality, input/output processing caused 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 is posted. Subsequent event control blocks (ECBs) are not posted.
- The ISAM check routine treats 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 is allowed (YES) or is not allowed (NO) while the STAE exit is executing.

ASYNCH=YES must be coded if:

- The STAE exit routine requests any supervisor services that require asynchronous interruptions to complete their normal processing.
- 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 develops when an asynchronous interruption occurs. Thus, it appears that the exit routine failed when, in reality, asynchronous exit handling caused 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.

Control returns to the instruction following the STAE macro instruction; 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, halt I/O, do not take asynchronous exits, transfer ownership 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.
b	One or more blanks must precede STAE.
STAE	
b	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 exception:

,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 you want to dynamically change the contents of the remote STAE parameter list, you can 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 is changed. The other field remains 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.
b	One or more blanks must precede STAE.
STAE	
b	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 exception:

,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, and 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

You can use the STATUS macro instruction to change the dispatchability status of one of your program's subtasks.

The STATUS macro instruction is also described in the *Supervisor Services and Macro Instructions*, with the exception of the SRB, ASID, and TASK parameters, which are restricted in use and available only to supervisor state, key zero callers. These restricted parameters allow the caller to manipulated the dispatchability of TCBs, SRBs, ASCBs, a STEP, or the SYSTEM.

The SYNCH operand of STATUS STOP is not supported in MVS/XA. Programs that issue STATUS STOP, SYNCH should be changed to issue STATUS STOP without the SYNCH operand. Users who specify the SYNCH operand with STATUS STOP will receive an MNOTE of severity 12 at assembly time.

Except for the TCB, all input parameters to this macro instruction can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

The description of the STATUS macro instruction is 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.
b	One or more blanks must precede STATUS.
STATUS	
b	One or more blanks must follow STATUS.

START STOP	
,TCB=*tcb addr* ,SRB ,SRB, ASID=*ASID addr* ,SRB,TASK=YES ,SRB,TASK=NO ,SRB,TASK=YES,ASID=*ASID addr* ,SRB,TASK=NO,ASID=*ASID addr*	*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. **Default:** TASK=YES
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are described as follows:

START
STOP

specifies that the appropriate START/STOP count is to be adjusted and the dispatchability bits are to be set/reset.

,TCB= *tcb addr*
,SRB
,SRB,ASID= *ASID addr*

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.

Note: The TCB resides in storage below 16 megabytes.

SRB — specifies that the STOP function affects the dispatchability of system-level SRBs only; all other tasks in the address space are set/reset nondispatchable. For START, the ASID addr specifies the address of a halfword containing the address space identifier.

TASK = — specifies whether the STATUS, STOP, and START functions affect the dispatchability of all other tasks in the address space. TASK=YES is the default. If TASK=YES is specified or defaulted, STATUS sets or resets task dispatchability in the address space. TASK=NO requests STATUS to ignore setting or resetting task dispatchability. TASK=NO modifies only system level SRB dispatchability and not TCB dispatchability. TASK=NO has the following restrictions:

- Issuers of STATUS must ensure that the dispatchability of all other tasks in the address space need not be modified.
- Issuers must be in key 0.
- Issuing programs must assemble the IHAASCB mapping macro into the user's program.

,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.

SET/RESET Options

The SET/RESET options of the STATUS macro instruction are written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede STATUS.
STATUS	
b	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 can 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 can only be specified with *tcb addr,(mask)*.
,ASID = *ASID addr*	*ASID addr:* RX-type address, or register (2) - (12). **Note:** For SET, this parameter can 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 as follows:

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.

,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). An address space that is exempt from system nondispatchability has bits ASCBXMPT or ASCBPXMT set to one. If you set the system ND or SD, you must not unconditionally request the CML lock of a non-exempt address space; if you do, a system deadlock could result.

STEP specifies that all job step TCBs (except the issuer's TCB) are to be set/reset nondispatchable.

tcb addr indicates that the specified TCB (except the issuer's 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.

Example 1

Operation: Set primary nondispatchability bit 3 for the specified TCB and all its subtasks.

```
STATUS SET,ND,TCBADDR,(3)
```

SUSPEND - Suspend Execution of a Request Block

The SUSPEND macro instruction places a request block (RB) in a suspended state until an expected event occurs, causing the task to resume processing.

The SUSPEND macro instruction is written as follows:

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

RB=PREVIOUS RB=CURRENT	**Default:** PREVIOUS

The parameters are explained as follows:

RB=PREVIOUS
RB=CURRENT
specifies which RB on the TCB to suspend. The previous RB is the caller's RB. The current RB is the first RB on the TCB chain.

The SUSPEND macro instruction uses registers as follows:

Register	Use	Contents after SUSPEND
0	TCB pointer	TCB address suspended
1	RB pointer	RB address suspended
2-10	Unused	Unchanged
11-13	Work registers	Unpredictable
14	Return address	Return address after SUSPEND
15	Work register	Unpredictable

Example 1

Operation: Suspend the execution of the most recently chained request block of the current task.

```
SUSPEND  RB=CURRENT
```

SVCUPDTE - SVC Update

The SVCUPDTE macro instruction provides a means to dynamically replace or delete SVC table entries. Callers who use this service are responsible for providing recovery. Improper deletion or replacement of MVS/XA provided SVC routines will cause unpredictable results and will probably cause a termination of the system.

The resource name, SYSVSVC TABLE, is available as the operand of an ENQ or DEQ macro, to be used when you must serialize the execution of a program that uses the SVCUPDTE macro. For information on using SYSVSVC TABLE, see Volume 1.

Users of this macro must be in supervisor state and key 0. Register 13 must contain the address of a 72-byte save area. Users of this macro must ensure that the code for the SVC routine added to the SVC table has the correct attributes for the type of SVC specified. See the topic "Modifying the SVC Table" in Volume 1 for additional information.

The SVCUPDTE macro is written as follows:

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

num	*num:* symbol, decimal number, hexadecimal number (for example, X'02'), or register (2) - (12). Do not specify *num* with extract.
,REPLACE ,DELETE ,EXTRACT	
,TYPE=1 ,TYPE=2 ,TYPE=3 ,TYPE=4 ,TYPE=5 ,TYPE=6	**Note:** This parameter is not valid with DELETE or EXTRACT.
,EP=*addr*	*addr:* A-type address, decimal number, Hexadecimal number (for example, X'FFEC00'), or register (2) - (12).
,EPNAME=*entry-name*	*entry-name:*symbol **Note:** EP and EPNAME are not valid with TYPE=5 and are not needed with the DELETE option.
,LOCKS=*(lname, lname,...)*	*lname:* CMS, DISP, SRM, LOCAL, or SALLOC. **Note:** LOCKS is invalid with DELETE and EXTRACT, and cannot be specified with TYPE=6.
,APF=YES ,APF=NO	**Default:** APF=NO **Note:** APF is not valid with DELETE.
,NPRMPT=YES ,NPRMPT=NO	**Default:** NPRMPT=NO **Note:** NPRMPT is not valid with DELETE.
,RELATED=*value*	*value:* any valid macro keyword specification.

The parameters are explained as follows:

num
specifies the number of the SVC that is being inserted or deleted.

,REPLACE
,DELETE
specifies the function to be performed. REPLACE indicates that a SVC table entry is to be inserted in the SVC table. This could be a new SVC or a replacement for an existing SVC. DELETE indicates that the specified SVC number is to be deleted from the SVC table. The SVCUPDTE routine deletes the number by placing the address of the SVC error routine into the table entry. When you execute an SVC instruction with a deleted SVC number, the result is an abnormal termination with an X'Fxx' abend. (xx is the

hexadecimal representation of the number specified.) However, if you issue an SVCUPDTE macro with a deleted SVC number, no abend results.

,TYPE=1
,TYPE=2
,TYPE=3
,TYPE=4
,TYPE=5
,TYPE=6

specifies the SVC type for a REPLACE request. See the topic "Program Conventions for SVC Routines" for information concerning the characteristics and restrictions for each type of SVC.

,EXTRACT

indicates that the user has supplied an EP or EPNAME and wishes to have the SVC number of that routine returned in register 0. The **num** parameter is not valid with this option.

,EP=*addr*

specifies the entry point address of the SVC routine. The addressing mode of the entry point is defined by bit 0 of the entry point address of the SVC routine. If bit 0=1, the SVC routine will be entered in 31-bit addressing mode; if bit 0=0, the SVC routine will be entered in 24-bit addressing mode.

,EPNAME=*entry-name*

specifies the entry name of the SVC routine. The entry name must be the load module name or alias of a module in LPA or the entry name of a module link edited into the nucleus.

Note: The service routine must obtain a 72-byte work area to support this option. The requestor of this service must not hold any lock higher than the VSMFIX lock.

,LOCKS=(*lname,lname,...***)**

specifies the lock(s) required when the SVC routine executes. The lock(s) specified can be any one or a combination of the following locks:

- CMS
- DISP
- SRM
- LOCAL
- SALLOC

Notes:

1. *TYPE=6 cannot specify any locks.*
2. *TYPE=1 must not specify LOCAL.*
3. *TYPE=3 and TYPE=4 must not specify SALLOC, and must not request Global Spin locks.*

,APF=YES
,APF=NO

specifies whether or not the SVC is to be APF-authorized.

,NPRMPT = YES
,NPRMPT = NO
indicates whether or not the SVC can be preempted for I/O interruptions.

,RELATED = *value*
provides information to document the macro by relating the function performed to another service or function. The format can be any valid coding value that the user chooses.

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

0 The macro completed successfully.

4. The macro instruction was coded incorrectly. For example, the user requested REPLACE without specifying an SVC number.

8 The DELETE parameter was not specified correctly.

0C A REPLACE request contained incorrect information. For example, the user specified an SVC type that was not 1 through 6.

10 A REPLACE request contained illogical information. For example:

- A type 5 SVC specified an entry point.
- A type 6 SVC specified a lock.
- A type 3 or type 4 SVC specified the DISP lock.
- Neither an entry point nor an EPNAME was provided for a REPLACE request that is not a type 5.
- Both an entry point and an EPNAME are provided.
- The entry point provided is zero.
- The CMS lock was requested without the LOCAL lock.

14 The function specified was not REPLACE, DELETE, or EXTRACT.

18 The user has attempted to update an extended SVC router entry in the SVC table.

1C Unable to locate the entry point address for an EPNAME specification.

20 An EXTRACT request contains illogical information. For example:

- Neither an entry point address nor an EPNAME is specified.
- Both an entry point address and an EPNAME are specified.
- An SVC number is specified.
- The entry point address specified is zero.

24 Unable to locate the SVC routine for the EXTRACT request.

28 An error occurred while updating the SVC table.

Example 1

Operation: Delete SVC 200 from the SVC table.

```
SVCUPDTE 200,DELETE
```

Example 2

Operation: Insert SVC 201 in the SVC table. This is a type 2 SVC, with entry point at location SVCADDR. The SVC cannot be preempted for I/O interruptions.

```
SVCUPDTE 201,REPLACE,NPRMPT=NO,TYPE=2,EP=SVCADDR
```

Example 3

Operation: Replace SVC 202 in the SVC table. This is a type 1 SVC with entry point at the location in register 2.

```
SVCUPDTE 202,REPLACE,TYPE=1,EP=(2)
```

Example 4

Operation: Replace SVC 203 in the SVC table. SVC 203 is a type 4 SVC requiring the LOCAL lock. The routine has been loaded into LPA with the name MYSVC.

```
SVCUPDTE 203,REPLACE,TYPE=4,LOCKS=LOCAL,EPNAME=MYSVC
```

Example 5

Operation: Determine the SVC number associated with the name IGC062. The SVC number is to be returned in register 0.

```
SVCUPDTE ,EXTRACT,EPNAME=IGC062
```

Example 6

Operation: Replace SVC 202 in the SVC table. This is a type 3 SVC with entry point at explicit location X'FFEC00'. Note that this example uses a symbol as the SVC number.

```
SVCUPDTE  SVCNUM,REPLACE,TYPE=3,EP=X'FFEC00'
          .
          .
          .
SVCNUM    EQU   202
```

SVCUPDTE (List Form)

The list form of the SVCUPDTE macro instruction builds a non-executable parameter list that can be referred to by the execute form of the SVCUPDTE macro.

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

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

num	*num:* symbol, decimal number, hexadecimal number (for example X'02'). **Note:** This parameter must be specified on the execute and the list form of the macro. Do not specify **num** with EXTRACT.
,REPLACE ,DELETE ,EXTRACT	
,TYPE=1 ,TYPE=2 ,TYPE=3 ,TYPE=4 ,TYPE=5 ,TYPE=6	**Note:** This parameter is not valid with DELETE.
,EP=*addr*	*addr:* A-type address, decimal number, or hexadecimal number (for example, X'FFEC00').
,EPNAME=*entry-name*	*entry-name:*symbol **Note:** EP and EPNAME are not valid with TYPE=5 and are not needed with the DELETE option. This parameter must be supplied on either the execute or the list form.
,LOCKS=*(lname, lname,...)*	*lname:* CMS, DISP, SRM, LOCAL, or SALLOC. **Note:** This option is not valid with DELETE or EXTRACT and must not be specified with TYPE=6.
,NPRMPT=YES ,NPRMPT=NO	**Default:** NPRMPT=NO **Note:** NPRMPT is not valid with DELETE.
,RELATED=*value*	*value:* any valid macro keyword specification.
,MF=L	

The parameters are explained under the standard form of the SVCUPDTE macro with the following exception:

,MF = L
specifies the list form of the SVCUPDTE instruction.

Example 1

Operation: Use the list form of the macro to replace SVC 202 in the SVC table. It is a type 2 SVC with entry point at location SVCADDR. The SVC routine needs the local lock.

```
SVCUPDTE 202,REPLACE,TYPE=2,LOCKS=LOCAL,MF=L,EP=SVCADDR
```

Example 2

Operation: Use the list form of the macro to replace SVC 201 in the SVC table. The routine is a type 2 SVC.

```
SVCUPDTE 201,REPLACE,TYPE=2,MF=L
```

SVCUPDTE (Execute Form)

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

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

num	*register (2) - (12).* **Note:** *This parameter must be supplied on either the execute or the list form of the macro with REPLACE or DELETE, and it must not be specified with EXTRACT.*
,EP=addr	*addr: register (2) - (12).* **Note:** *This parameter is not valid with TYPE=5 and must be supplied on either the execute or the list form of the macro. This parameter is not needed with the delete option.*
,RELATED=value	*value: any valid macro keyword specification.*
,MF=(E, addr)	*addr: RX-type address or register (2) - (12).*

The parameters are explained under the standard form of the SVCUPDTE macro instruction with the following exception:

,MF=(E,*addr*)
specifies the execute form of the SVCUPDTE instruction.

Example 1

Operation: Use the execute form of the SVCUPDTE macro instruction to perform the function specified by the remote control parameter list whose address is given in register 2.

```
SVCUPDTE MF=(E,(2))
```

SWAREQ - Invoke SWA Manager in Locate Mode

The SWAREQ macro has no standard form. It only has a list, an execute, and a modify form. The MF parameter, which indicates the form of the macro, is required.

When you invoke this macro in execute form, it uses the two parameters, FCODE and EPA, to modify the parameter list, which is at the location you specify by the **addr** value in the **MF=(E,addr)** parameter. After ensuring the validity of the parameters, it invokes the SWA manager in locate mode. The SWA manager obtains its input from the parameter list, and performs the function accociated with the specified FCODE. If you do not specify any parameters, the macro assumes the parameter list already exists, and it simply invokes the SWA manager.

The modify form of SWAREQ is functionally the same as the execute form, except that the macro only modifies the parameter list without invoking the SWA manager. The list form of SWAREQ generates the parameter list that is modified by the other two forms of the macro, and it does not invoke the SWA manager.

The list form of the SWAREQ macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede SWAREQ.
SWAREQ	
b	One or more blanks must follow SWAREQ.
,FCODE=*fncde*	*fncde:* function code
,EPA=*addr*	*addr:* external parameter area pointer address. In the list form, this address may only be specified symbolically.
,MF=L	

The parameters are explained as follows:

,FCODE=*fncde*
specifies the function code for the locate mode request. Valid codes are:

AC Assign/Conditional
AL Assign/Locate
DB Delete block
LA Locate/All
RL Read/Locate
WL Write/Locate

For more information about the meaning of each code, see volume 1 of this book.

,EPA=*addr*
specifies the address of the EPA pointer.

,MF=L
specifies the list form of the SWAREQ macro instruction.

SWAREQ (Execute Form)

The execute form of the SWAREQ macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede SWAREQ.
SWAREQ	
b	One or more blanks must follow SWAREQ.
,FCODE=*fncde*	*fncde:* function code
,EPA=*addr*	*addr:* external parameter area pointer address. It may be specifed symbolically, as a register enclosed in parentheses, or as a symbol equated to a register enclosed in parentheses.
,MF=*(E, addr)*	*addr:* RX-type address or register (1) - (12).

The parameters are explained under the list form of the SWAREQ macro instruction, with the following exceptions:

,MF=(E,addr)
E specifies the execute form of the SWAREQ macro instruction, and **addr** specifies the address of the parameter list.

SWAREQ (Modify Form)

The modify form of the SWAREQ macro is written as follows:

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

,FCODE=*fncde*	*fncde:* function code
,EPA=*addr*	*addr:* external parameter area pointer address. It may be specifed symbolically, as a register enclosed in parentheses, or as a symbol equated to a register enclosed in parentheses.
,MF=*(M,addr)*	*addr:* RX-type address or register (1) - (12) .

The parameters are explained under the list form of the SWAREQ macro instruction, with the following exceptions:

,MF=(M,addr)
M specifies the modify form of the SWAREQ macro instruction, and **addr** specifies the address of the parameter list.

SYMREC - Process Symptom Record

The SYMREC macro updates the symptom record with system environment information and then logs the symptom record in the SYS1.LOGREC data set. The symptom record is a data area in the user's application that has been mapped by the ADSR macro and that is referenced by a parameter of the SYMREC macro. The data in the symptom record is a description of a programming failure and a description of the environment in which the failure occurred. As the application detects errors during execution, it stores diagnostic information into the symptom record and issues SYMREC to log the information.

The caller must be enabled for interrupts. Suspend locks may be held, but spin locks cannot be held. If the SYMREC request is issued in disabled mode, the record is not written and a return code of X'0C' and a reason code of X'144' are provided.

The caller must not be executing in secondary addressing mode. That is, bit 16 of the PSW must be zero. The SYMREC service routine is invoked via the PC (Program Call) instruction. If the SYMREC request is issued in secondary addressing mode, the request abnormally terminates when the PC instruction is executed. This abend occurs before the SYMREC service can establish recovery to intercept the PC failure. While the SYMREC macro can be issued in 24-bit or 31-bit addressing mode, the addresses passed to the SYMREC service must be 31-bit addresses. The service routine uses the addresses as passed.

When SYMREC is invoked, it checks that all the required input fields of the ADSR symptom record are set by the caller. If the required input fields are not set, SYMREC issues appropriate return and reason codes (described in *SPL: System Macros and Facilities, Volume 1*).

The SYMREC macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede SYMREC.
SYMREC	
b	One or more blanks must follow SYMREC.
SR=*addr*	*addr:* A-type address or register (2) - (12).

The parameters are explained as follows:

SR=*addr*
specifies the address of the symptom record. The SR keyword is required.

SYMREC (List Form)

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede SYMREC.
SYMREC	
b	One or more blanks must follow SYMREC.
SR=*addr*	*addr:* A-type address (31 bit).
,MF=(L)	

The parameters are explained under the standard form of the SYMREC macro instruction with the following exception:

,MF=L
specifies the list form of the SYMREC macro instruction.

SYMREC (Execute Form)

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

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

SR = *addr*	*addr:* A-type address (31 bit) or register (2) - (12).
,MF = *(E,list addr)*	*list addr:* RX-type address or register (2) - (12).

The parameters are explained under the standard form of the SYMREC macro instruction with the following exception:

,MF = (E,*list addr)*
specifies the execute form of the SYMREC macro instruction. This form uses a remote parameter list.

SYNCH - Take a Synchronous Exit to a Processing Program

If you are executing in 31-bit addressing mode, you must use the MVS/XA version of this macro instruction.

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 also described in *Supervisor Services and Macro Instructions* with the exception of the KEYADDR, STATE, and KEYMASK parameters. These parameters are restricted in use to programs in supervisor state, key 0-7, or APF-authorized.

On entry to the processing program, the high order bit, bit 0, of register 14 is set to indicate the addressing mode of the issuer of the SYNCH macro. If bit 0 is 0, the issuer is executing in 24-bit addressing mode; if bit 0 is 1, the issuer is executing in 31-bit addressing mode.

The SYNCH macro instruction is written as follows:

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

entry point addr	*entry point addr:* RX-type address, or register (2) - (12) or (15).
,RESTORE=NO ,RESTORE=YES	**Default:** RESTORE=NO
,KEYADDR=*addr* ,KEYADDR=NOKEYADDR	*addr:* RX-type address, or register (2) - (12) **Default:** KEYADDR=NOKEYADDR (The key in the TCB is used.)
,STATE=PROB ,STATE=SUPV	**Default:** STATE=PROB
,KEYMASK=*addr*	*addr:* RX-type address, or register (0) - (12).
,AMODE=24 ,AMODE=31 ,AMODE=DEFINED ,AMODE=CALLER	**Default:** AMODE=CALLER **Note:** AMODE=DEFINED can only be specified if the entry point is provided in a register.

The parameters are explained as follows:

entry point addr
specifies the address of the entry point of the processing program to receive control.

,RESTORE=NO
,RESTORE=YES
specifies whether registers 2-13 are to be restored when control is returned to the issuer of SYNCH.

,KEYADDR=*addr*
,KEYADDR=NOKEYADDR
addr specifies the address of a one-byte area that contains the key in which the exit is to receive control. The key must be in bits 0-3; bits 4-7 must be zero. If KEYADDR=*addr* is not specified, the key in the TCB is used as the default.

,STATE=PROB
,STATE=SUPV
specifies the state in which the requested program receives control. PROB specifies problem state and SUPV specifies supervisor state.

,KEYMASK=*addr*
specifies the address of a halfword to be ORed with the TCBPKM (which already might have been modified by KEYADDR) to produce the PKM of the routine to which the synchronous exit is to be taken.

,AMODE=24
,AMODE=31
,AMODE=DEFINED
,AMODE=CALLER
specifies the addressing mode in which the requested program is to receive control.

If AMODE=24 is specified, the requested program will receive control in 24-bit addressing mode.

If AMODE=31 is specified, the requested program will receive control in 31 bit addressing mode.

If AMODE=DEFINED is specified, the user must provide the entry point using a register and not an RX-type address. The requested program will receive control in the addressing mode indicated by the high order bit of the entry point address. If the bit is off, the requested program will receive control in 24-bit addressing mode; if the bit is set, the requested program will receive control in 31-bit addressing mode.

If AMODE=CALLER is specified, the requested program will receive control in the addressing mode of the caller.

Example 1

Operation: Take a synchronous exit to a processing program whose entry point address is specified in register 8.

```
SYNCH (8)
```

Example 2

Operation: Take a synchronous exit to a processing program labeled SUBRTN and restore registers 2-13 when control is returned.

```
SYNCH SUBRTN,RESTORE=YES
```

Example 3

Operation: Take a synchronous exit to a processing program whose entry point address is specified in register 5, modify the program's TCBPKM by the KEYADDR and KEYMASK values, and restore registers 2-13 when control returns.

```
SYNCH (5),RESTORE=YES,KEYADDR=KEYBYTE,KEYMASK=MSKADDR
   .
   .
   .
KEYBYTE DC X'80'
MSKADDR DC X'0080'
```

Example 4

Operation: Take a synchronous exit to the program located at the address given in register 8 and restore registers 2-13 when control returns. Indicate that this program is to execute in 24-bit addressing mode.

```
SYNCH (8),RESTORE=YES,AMODE=24
```

SYNCH (List Form)

The list form of the SYNCH macro instruction is used to construct a control program parameter list.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede SYNCH.
SYNCH	
b	One or more blanks must follow SYNCH.
,RESTORE=NO ,RESTORE=YES	**Default:** RESTORE=NO
,STATE=PROB ,STATE=SUPV	**Default:** STATE=PROB
,KEYMASK=*addr*	*addr:* A-type address.
,AMODE=24 ,AMODE=31 ,AMODE=DEFINED ,AMODE=CALLER	**Default:** AMODE=CALLER
,MF=L	

The parameters are explained under the standard form of the SYNCH macro instruction with the following exception:

,MF=L
specifies the list form of the SYNCH macro instructions.

Example 1

Operation: Use the list form of the SYNCH macro instruction to specify that registers 2-13 are to be restored when control returns from executing the SYNCH macro instruction and that the addressing mode of the program is to be defined by the high-order bit of the entry point address. Assume that the execute form of the macro instruction specifies the program address.

```
SYNCH  ,RESTORE=YES,AMODE=DEFINED,MF=L
```

SYNCH (Execute Form)

The execute form of the SYNCH macro instruction uses a remote control program parameter list that can be generated by the list form of SYNCH.

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

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

entry point addr	*entry point addr:* RX-type address, or register (2) - (12) or (15).
,RESTORE=NO ,RESTORE=YES	**Default:** RESTORE=NO
,KEYADDR=*addr* ,KEYADDR=NOKEYADDR	*addr:* RX-type address, or register (2) - (12).
,STATE=PROB ,STATE=SUPV	**Default:** STATE=PROB
,KEYMASK=*addr*	*addr:* RX-type address, or register (0) - (12).
,AMODE=24 ,AMODE=31 ,AMODE=DEFINED ,AMODE=CALLER	**Default:** AMODE=CALLER **Note:** AMODE=DEFINED can only be specified if the entry point is provided in a register.
,MF=(E, *ctrl addr*)	*ctrl addr:* RX-type address or register (1), (2) - (12)

The parameters are explained under the standard form of the SYNCH macro instruction with the following exceptions:

,KEYADDR=NOKEYADDR
indicates that the default(the key in the TCB) should be used instead of the key in the parameter list defined by a list form of the macro instruction.

,MF=(E,*ctrl addr***)**
specifies the execute form of the SYNCH macro instruction using a list generated by the list form of SYNCH.

Example 1

Operation: Use the execute form of the SYNCH macro instruction to take a synchronous exit to the program located at the address given in register 8 and restore registers 2-13 when control returns. Indicate that the program is to receive control in the same addressing mode as the caller and that the parameter list is located at SYNCHL2.

```
SYNCH (8),RESTORE=YES,AMODE=CALLER,MF=(E,SYNCHL2)
```

SYSEVENT - System Event

The SYSEVENT macro instruction provides the interface to the system resource manager (SRM). Through the use of SYSEVENT mnemonics, you can notify SRM of an event or request SRM to perform a specific function.

You must include the CVT mapping macro as a DSECT in the calling program. If a specific SYSEVENT requires additional parameters, you must load register 1 with the address of a parameter list before issuing the macro instruction.

Callers who use ENTRY=BRANCH must:

- Be in supervisor state, key 0
- Hold no locks higher than the SRM lock
- Provide the address of a serialized 72-byte save are in register 13

Callers who use ENTRY=SVC must:

- Be APF authorized, supervisor state, or key 0

Note: You do not require any authorization to issue SYSEVENT REQSERVC.

Additional restrictions concerning the use of each SYSEVENT, including input and output requirements, are given following the description of the parameters.

Only the SYSEVENTs that are commonly used by subsystems or other external users are documented here. The other SYSEVENTs are used internally by the MVS system and are not included in this documentation. Refer to the "System Resource Manager" section of the *System Logic Library* for a complete list of the SYSEVENTs along with information concerning the internal interfaces for the mnemonics. *System Programming Library: Initialization and Tuning* provides additional information concerning the input, output, and environment for issuing the SYSEVENTs. The *Debugging Handbook* contains a summary of the SYSEVENTs.

The SYSEVENT macro instruction is written as follows:

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

sysevent mnemonic	*sysevent mnemonic:* symbol. **Note:** See the description of the parameters for the valid options.
,ENTRY=SVC ,ENTRY=BRANCH	**Defaults:** ENTRY=BRANCH for the following SYSEVENTs: TRAXERPT TRAXFRPT TRAXRPT HOLD NOHOLD ENTRY=SVC for the following SYSEVENTs: COPYDMDT DONTSWAP REQPGDAT OKSWAP REQSERVC TRANSWAP
,TYPE=INTERVAL	**Note:** The TYPE parameter is valid and required only for SYSEVENT REQPGDAT.

The parameters are explained as follows:

sysevent mnemonic
identifies the SYSEVENT being requested. The SYSEVENT service routine inserts the code corresponding to the mnemonic into byte 3 of register 0.

,ENTRY=SVC
,ENTRY=BRANCH
specifies the type of interface to SRM (SVC or branch).

Only users who do not hold a lock can specify ENTRY=SVC. This is the default for DONTSWAP, OKSWAP, TRANSWAP, COPYDMDT, REQPGDAT, and REQSERVC.

ENTRY=BRANCH is required if the caller holds a lock and for all "fast path" SYSEVENTs. The "fast path" SYSEVENTs include HOLD, NOHOLD, TRAXERPT, TRAXFRPT, and TRAXRPT. ENTRY=BRANCH is the default for these "fast path" SYSEVENTs. For branch entry, callers must provide a 72-byte save area and place the address of the save area in register 13.

,TYPE=INTERVAL
indicates that this is an interim measurement and the paging data is not to be reset. This keyword is required and can be used only with the REQPGDAT SYSEVENT.

SYSEVENT mnemonics

A description of the SYSEVENTs available for restricted use follows. These mnemonics are grouped according to the basic function that they perform.

Notify SRM of Transaction Completion

The SYSEVENTs TRAXRPT, TRAXFRPT, and TRAXERPT notify SRM that a subsystem transaction has completed and provide the transaction's starting time or elapsed time and, optionally, its resource utilization. This performance data can be reported using the resource management facility (RMF).

To obtain reports, an IEAICSxx parmlib member must be in effect and RMF workload activity reporting must be active. See *System Programming Library: Initialization and Tuning* for additional information concerning the IEAICSxx parmlib member.

In addition to the general requirements for SYSEVENTs, TRAXRPT, TRAXFRPT, and TRAXERPT require the user to:

- Provide a parameter list
- If the issuing program is disabled, ensure that the parameter list and save area are fixed
- Provide error recovery

A description of the individual mnemonics follows:

TRAXRPT
notifies SRM that a transaction has completed and provides its start time. Register 1 must point to a serialized parameter list with the following format:

Offset in Hex	Length	Field Description
00	8	Transaction start time in store clock instruction (STCK) format
08	8	Subsystem name
10	8	Transaction name or blanks
18	8	User identification or blanks
20	8	Transaction class or blanks

Note: You can obtain this parameter list by using the IHATRBPL mapping macro in your program.

The names must be in ECBDIC format, left-justified, and padded with blanks. Note that the subsystem name is restricted to four characters in length.

TRAXFRPT

notifies SRM that a transaction has completed and provides the elapsed time. Because the issuer calculates the elapsed time before issuing the macro instruction, this path is shorter than the path for TRAXRPT. Register 1 must point to a serialized parameter list with the following format:

Offset in Hex	Length	Field Description
00	4	Transaction elapsed time (1.024 milliseconds units)
04	4	Reserved - must be zero
08	8	Subsystem name
10	8	Transaction name or blanks
18	8	User identification or blanks
20	8	Transaction class or blanks

Note: You can obtain this parameter list by including the mapping macro IHATRBPL in your program. The names must be in EBCDIC format, left-justified, and padded with blanks. Note that the subsystem name is restricted to four characters in length.

TRAXERPT

notifies SRM that a transaction has completed, provides its start time, and includes resource utilization data for determining service consumption. Register 1 must point to a serialized parameter list in the following format:

Offset in Hex	Length	Field Description
00	8	Transaction start time in store clock assembler instruction (STCK) format
08	8	Subsystem name
10	8	Transaction name or blanks
18	8	User identification or blanks
20	8	Transaction class or blanks
28	8	Task (TCB) time in store clock assembler instruction (STCK) format or zeros
30	8	SRB time in store clock instruction (STCK) format or zeros
38	8	Main storage occupancy in page seconds (pages times msec, where msec is task (TCB) time in 1.024 millisecond units)
40	4	Logical I/O count or zeros
44	1	X'00' if the previous field contains the logical I/O count X'80' if the previous field contains the device connect time interval (DCTI)
45	3	Reserved must be zero

Note: You can obtain this parameter list by including the mapping macro IHATREPL in your program.

The names must be in EBCDIC format, left-justified, and padded with blanks. Note that the subsystem name is restricted to four characters in length.

When SYSEVENT processing is completed, the subsystem regains control at the instruction following the SYSEVENT macro instruction. Register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Data for the transaction has been reported successfully to the SRM.
08	Processing could not be completed at this time. No queue elements are available for recording the data. No data is reported to the SRM, but an immediate reissue could be successful.
0C	Reporting is temporarily suspended for one of the following reasons: - RMF workload activity reporting is not active. - There is no installation control specification (IEAICSxx parmlib member with RPGN specified for some subsystem other than TSO) in effect. The TOD clock is stopped. No data is reported, but a later reissue could be successful.
10	Reporting is inoperative. The TOD clock is in error or the reporting interface is not installed. No data is reported.

Example 1

Operation: Use the SYSEVENT TRAXRPT to report transaction data providing transaction identifiers and the transaction start time.

```
  .
  .
  .
Transaction begins      Initialize transaction identifiers
       (TRAXDES)
 STCK INITTIME          Save start time
  .
Process transaction
Transaction completes
 LA R13,SVAREA          Provide 72-byte save area
 LA R1,PARMS            Point to parameter area
 MVC 0(8,R1),INITTIME   Move in start time
 MVC 8(32,R1),TRAXDESC  Get subsystem name, transaction
                        name, userid, and class
 SYSEVENT TRAXRPT
  .
  .
INITTIME DS D
PARMS    DS 5D
SVAREA   DS 18F
TRAXDESC DS CL32
```

Example 2

Operation: Use the SYSEVENT TRAXERPT to report transaction data, providing transaction identifiers, start time and resource utilization data.

```
 .
 .
 .
Transaction begins      Initialize transaction identifiers
       (TRAXDESC)
 STCK INITTIME          Save start time
 .
Process transaction     Accumulate resource utilization data
       (TRAXDESC)
 .
Transaction completes
 LA  R13,SVAREA         Provide 72-byte save area
 LA  R1,PARMS           Point to parameter area
 MVC 0(8,R1),INITTIME   Move in start time
 MVC 8(64,R1),TRAXDESC  Get subsystem name, transaction
                        name, user id, class, and
                        resource utilization data
 SYSEVENT TRAXERPT
 .
 .
 .
INITTIME DS D
PARMS    DS 9D
SVAREA   DS 18F
TRAXDESC DS CL64
```

Example 3

Operation: Use the SYSEVENT TRAXFRPT to report transaction data, providing transaction identifiers and calculating the elapsed time.

```
 .
 .
 .
Transaction begins      Initialize transaction identifiers
       (TRAXDESC)
 .
 .
Process transaction     Calculate elapsed time (TOTLTIME)
 .
Transaction completes   Calculate elapsed time (TOTLTIME)
 LA  R13,SVAREA         Provide 72-byte save area
 LA  R1,PARMS           Point to parameter area
 MVC 0(4,R1),TOTLTIME   Move in elapsed time
 XC  4(4,R1),4(R1)      Clear reserved field
 MVC 8(32,R1),TRAXDESC  Get subsystem name, transaction name,
                        user id, and class
 SYSEVENT TRAXFRPT
 .
 .
 .
TOTLTIME DS F
PARMS    DS 5D
SVAREA   DS 18F
TRAXDESC DS CL32
```

Control Swapping

The SYSEVENTs HOLD, NOHOLD, DONTSWAP, OKSWAP, and TRANSWAP control swapping. The choice of mnemonic depends on the period of time for which the address space is to be non-swappable.

For a very short period of time (a path length of less than 200 instructions with no waits, SVCs, or I/O between the HOLD and NOHOLD SYSEVENTs), use HOLD to make the address space non-swappable and NOHOLD to make it swappable.

For a short period of time (less than one minute), use DONTSWAP to make it non-swappable and OKSWAP to make it swappable.

For an extended period of time (indefinite time span), use TRANSWAP to make the address space non-swappable and OKSWAP to make it swappable.

A description of the individual mnemonics follows:

HOLD
notifies SRM that the address space that issued this SYSEVENT cannot be swapped out until the address space issues a NOHOLD SYSEVENT and the HOLD count in the SRM user control block (OUCB) is zero. SRM increases the HOLD count in the OUCB each time a program in the address space issues SYSEVENT HOLD.

You should use this SYSEVENT to prevent a swap out while holding a critical resource for a very short period of time. If you are in SRB mode, you must issue SYSEVENT NOHOLD before SRB processing completes. SYSEVENT HOLD does not require any input parameters.

NOHOLD
notifies SRM that the current address space, from which a program has issued SYSEVENT HOLD, will be considered for swapping when the HOLD count in the OUCB reaches zero. SRM decreases this HOLD count when a program in the address space issues SYSEVENT NOHOLD.

DONTSWAP
notifies SRM that the address space from which this SYSEVENT is issued cannot be swapped out until a program issues SYSEVENT OKSWAP from the address space or MVS issues INITATT or INITDET.

No input parameters are required. One of the following codes will be returned in register 1, byte 3:

Hexadecimal Code	Meaning
00	The request was honored.
04	The request was not honored because it was not for the current address space.
08	The request was not honored because the issuer was not authorized or the outstanding count of DONTSWAP requests had reached its maximum.

OKSWAP

notifies SRM that the address space from which the SYSEVENT was issued can be considered for swapping.

No input parameters are required. One of the following codes will be returned in register 1, byte 3:

Hexadecimal Code	Meaning
00	The request was honored.
04	The request was not honored because it was not for the current address space.
08	The request was not honored because the issuer was not authorized.

TRANSWAP

forces a swap out. After the subsequent swap-in, frames are allocated from preferred storage and the address space is marked non-swappable. TRANSWAP prevents programs from allocating frames in reconfigurable storage. If the request is for the current address space and a dependency on the completion of the transaction exists, register 1 can contain the address of an ECB to be posted when the address space is swapped out.

One of the following codes will be returned in register 1, byte 3:

Hexadecimal Code	Meaning
00	The request was honored.
04	The transition was previously done.

If an ECB was specified, the following POST codes may occur:

Hexadecimal Code	Meaning
00	The transition is complete.
04	The address space became non-swappable before it could be swapped out.

Example 1

Operation: Make the current address space non-swappable for a path length of less than 200 instructions that contains no waits, SVCs or I/O.

```
         LA   R13,SVAREA
         SYSEVENT HOLD
             .
             .
             .
         LA   R13,SVAREA
         SYSEVENT NOHOLD
             .
             .
             .
SVAREA   DS   18F
```

Example 2

Operation: Make the current address space non-swappable for a time period of less than one minute.

```
SYSEVENT DONTSWAP
         .
         .
         .
SYSEVENT OKSWAP
```

Example 3

Operation: Make the current address space non-swappable for an indefinite period of time.

```
SYSEVENT TRANSWAP
         .
         .
         .
```

Obtain System Measurement Information

SYSEVENTs REQPGDAT, REQSERVC, and COPYDMDT are used to obtain system measurement information. REQPGDAT and REQSERVC provide information about a particular address space; COPYDMDT provides information about the entire system.

The user must supply the address of a data area large enough to store the requested data.

A description of the individual mnemonics follows:

REQPGDAT
indicates a request for user-paging data. Callers must specify TYPE=INTERVAL when using this SYSEVENT. If you do not specify TYPE=INTERVAL with REQPGDAT, the accounting information is reset to zero.

On entry, register 1 must contain the address of a fixed sixteen-word area where the service data is to be stored. On return, this area contains:

Offset in Hex	Length	Field Description
00	4	Count of non-VIO page-ins
04	4	Count of non-VIO page-outs
08	4	Count of non-VIO reclaims
0C	4	Count of VIO page-ins
10	4	Count of VIO page-outs
14	4	Count of VIO reclaims
18	4	Count of pages swapped in
1C	4	Count of pages swapped out
20	4	Count of swap-outs
24	4	Count of common area page-ins
28	4	Count of common area reclaims
2C	4	Count of pages stolen
30	4	Count of LPA page-ins
34	4	Count of LPA reclaims
38	8	Count of CPU page-seconds

On return, register 15, byte 3 contains the following information:

Hexadecimal Code	Meaning
00	Successful return
04	Unsuccessful return

REQSERVC

obtains service-related data from SRM for a given address space. SYSEVENT REQSERVC must be issued before the address space data is reset at job termination.

Register 1 must contain the address of a three-word area where the service data is to be stored. When the SYSEVENT completes, this area contains the information specified according to whether the address space is or is not a TSO address space.

For a TSO address space:

Word	Contents
1	Total service for the job
2	Total transaction active time in 1.024 millisecond units of time
3 (Bytes 0-1)	Performance group number last assigned to the address space
3 (Bytes 2-3)	Total number of transactions

In the case of a non-TSO address space:

Word	Contents
1	Total service for the session
2	Total active time for all transactions in 1.024 millisecond units of time
3 (Bytes 0-1)	Performance group number last assigned to the address space
3 (Bytes 2-3)	Zeros

On return register 15, byte 3 contains:

Hexadecimal Code	Meaning
00	Data obtained
04	Data lost because of an accumulation control block error

COPYDMDT

requests the list of SRM related parmlib members that are currently in effect. The user can also request a copy of SRM's domain table (DMDT).

On entry, register 1 must contain the address of a fixed data area. If the user requested a copy of the DMDT, the caller must enter X'40' in the first byte of the fixed data area

(parameter list). The data area must be large enough to contain the following information:

Offset in Hex	Length	Field Description
00	1	X'80' indicates that non-swappable address spaces were counted in the CMPLs of domains X'40' indicates that the domain table is to be included.
04	4	Reserved
08	8	The time when the displayed values were in effect
10	8	IPS parmlib member name
18	8	OPT parmlib member name
20	8	Installation control specification parmlib member name
28	2	Count of domains
2A	Variable	Domain table

Example 1

Operation: Obtain user-paging data for the current address space.

```
         LA   1,PAGDATA
         SYSEVENT REQPGDAT,TYPE=INTERVAL
                .
                .
                .
PAGDATA  DS   16F
```

Example 2

Operation: Obtain service-related data for the current address space.

```
         LA   1,SERVDATA
         SYSEVENT REQSERVC
                .
                .
                .
SERVDATA DS   3F
```

Example 3

Operation: List SRM's related parmlib members that are in effect. Do not list the domain table.

```
         LA   1,DOMPARM
         SYSEVENT COPYDMDT
              .
              .
              .
DOMPARM  DS   10F
```

TCTL - Transfer Control from an SRB Process

The TCTL (transfer control) macro instruction allows an SRB process to exit from its processing and to pass control directly to a task without going through the dispatcher.

The TCTL macro instruction is coded as follows:

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

TCB=(4) TCB=*tcbaddr*	**Default:** TCB address contents of register (4) *tcbaddr:* A-type address or registers (2) - (12).

The parameters are explained as follows:

TCB=(4)
TCB=*tcbaddr*
specifies the task designated for dispatching. Register (4) is the default; it is assumed to contain the appropriate TCB address.

Note: The TCB resides in storage below 16 megabytes.

The TCTL macro instruction uses registers as follows:

Register	Use
0-3	Work registers
4	TCB address
5-14	Work registers
15	EPA of the TCTL routine

Example 1

Operation: From SRB mode processing, terminate the SRB and give control to the task specified in register 4.

```
TCTL TCB=(4)
```

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.
b	One or more blanks must precede TESTAUTH.
TESTAUTH	
b	One or more blanks must follow TESTAUTH.

FCTN=*fctn* FCTN=*fctn*,AUTH=*auth*	*fctn:* decimal digit 0 or 1 or register (2) - (12). *auth:* decimal digit 0 or 1, or register (2) - (12). **Default:** FCTN=0
,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 as follows:

FCTN=*fctn*
FCTN=*fctn* **,AUTH=***auth*
specifies the authorization (via APF) of a program.

FCTN=0 specifies that APF-authorization is not checked.

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 FCTN=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. In MVS/XA, AUTH is generally not coded except in a testing environment.

,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.)

Note: TESTAUTH is used to test one or more of three conditions FCTN,STATE, or KEY. If any of the requested conditions are tested favorably, a return code of 0 is returned in register 15. If all of the requested conditions are tested unfavorably, the return code is set to 4.

,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). If BRANCH=YES is specified, registers 2 and 3 are modified by the TESTAUTH routine.

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 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
```

T6EXIT - Type 6 Exit

The T6EXIT macro instruction returns control from a Type 6 SVC routine to the SVC first level interrupt handler (FLIH). This exit macro can only be used in a Type 6 SVC.

The T6EXIT macro instruction is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede T6EXIT.
T6EXIT	
b	One or more blanks must follow T6EXIT.

RETURN=CALLER	**Default:** RETURN=CALLER
RETURN=DISPATCH	
RETURN=SRB	

The explanation of the RETURN parameter is as follows:

RETURN=
specifies how the Type 6 SVC has chosen to exit.

CALLER specifies that the return is directly to the caller or issuer of the SVC without going through the dispatcher. CALLER is the default return option.

DISPATCH specifies that the return should be through the dispatcher. This function is for the use of routines that have suspended the current task.

No registers are returned to the caller.

SRB specifies that the SVC FLIH should immediately dispatch an SRB. This SRB must:

- Be initialized by the Type 6 SVC
- Be pointed to by register 1
- Execute in the same address space as the SVC. The SRB has the same format as the SCHEDULE SRB.

Note: No registers are returned to the caller.

Example 1

Operation: Terminate Type 6 SVC processing and return control from the Type 6 SVC to the caller of the SVC.

```
T6EXIT  RETURN=CALLER
```

VRADATA - Update Variable Recording Area Data

The VRADATA macro instruction copies service information into a variable recording area (VRA), usually the system diagnostic work area (SDWAVRA). This information can later be written to the SYS1.LOGREC data set if software errors occur. (See the SETRP macro instruction, RECORD=YES keyword description, for more information on recording the SDWA control block.) The information copied into the VRA using this macro instruction is in a key, length, data format defined by the IHAVRA mapping macro instruction. The key and length are one byte fields; the data can vary in length. The IHAVRA mapping macro is shown in the *Debugging Handbook*.

The VRADATA macro instruction is written as follows:

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

VRAINIT=*vra addr*	*vra addr:* RX-type address, or the symbol 'SDWAVRA'. **Default:** address of SDWAVRA
,VRACLEN=*curr len addr* or *(curr len addr,0)* ,VRAMLEN=*max len addr*	*curr len addr:* RX-type address. **Default:** address of SDWAURAL. *max len addr:* RX-type address. **Default:** address of SDWAVRAL.
,KEY=*key nmbr*	*key nmbr:* symbol or decimal digit.
,LENADDR=*data len addr* ,LEN=*data len value*	*data len addr:* RX-type address. *data len value:* symbol or decimal digit. **Default:** length of DATA storage.
,DATA=*data addr*	*data addr:* RX-type address, or register (1) - (15).
,SDWAREG=*(reg, descr)*	*reg:* symbol or decimal digits 1-15. **Default:** 1 *descr:* SET or NOTSET
,VRAREG=*(reg,descr)*	*reg:* symbol or decimal digits 1-15. **Default:** 14 *descr:* SET or NOTSET **Default:** NOTSET if VRAINIT is specified, otherwise SET.
,WORKREG=*reg*	*reg:* symbol or decimal digits 1-15. **Default:** 15
,TYPE=*(lentype,testtype)*	*lentype:* LEN or NOLEN or NOL *testtype:* TEST or NOTEST or NOT **Default:** LEN, TEST

The parameters are explained as follows:

,VRAINIT = *vra addr*
specifies the address of the variable recording area to be initialized and updated. The value in the register specified by the VRAREG keyword is also initialized unless VRAREG=(,SET) is specified. If VRAINIT=SDWAVRA is specified, the SDWA control block is also updated to indicate that the VRA contains hexadecimal data, and data in key-length-data format. If VRAINIT is not specified, VRAINIT=SDWAVRA is assumed.

All subsequent VRADATA macro instructions use the specified VRAINIT value until you specify another VRAINIT value.

,VRACLEN = *curr len addr*
specifies the address of a one-byte field that contains the length of the current VRA. This value changes as information is added in the VRA. If you do not specify VRACLEN, you can obtain the current length of the VRA from the SDWAURAL field of the SDWA.

,VRACLEN = *(curr len addr, 0)*
specifies that the area containing the length is to be zeroed.

All subsequent VRADATA macro instructions use the specified VRACLEN value until you specify another VRACLEN value.

,VRAMLEN = *max len addr*
specifies the address of a two-byte field that contains the maximum length of the VRA. If you do not specify VRAMLEN, the maximum length is obtained from SDWAVRAL.

All subsequent VRADATA macro instructions use the specified VRAMLEN value until you specify another VRAMLEN value.

,KEY = *key number*
specifies the key value to be placed in the VRAKEY field of the current VRA entry. The IHAVRA mapping macro defines the valid key values.

,LENADDR = *data len addr*
,LEN = *data len*
specifies the length of the data for the VRA entry. The maximum length is 255 bytes. Omit this keyword unless the DATA keyword is a register value or a displacement plus a register, or if the defined data length must be overridden because it is larger than 255 bytes. For bit string data, use this keyword to indicate how many bytes the bit string occupies. The data length field pointed to by LENADDR must be a two-byte area with the length right-justified in the area.

,DATA = *data addr*
specifies the address of the data to be copied into the VRA. The data must correspond to the key specified by the KEY parameter. If you specify DATA, you must specify KEY. You must also specify LEN or LENADDR if DATA has a register value or if the data length is greater than 255 bytes.

,SDWAREG=*reg*

specifies a register containing the address of the SDWA control block. You must place the address in this register before invoking VRADATA. THE VRADATA macro instruction preserves the contents of this register. If you do not specify SDWAREG, register 1 is the default.

,VRAREG=*(reg,descr)*

specifies a register to contain the address of the next available space in the VRA and a description of whether or not the register value is already set (SET) or not set (NOTSET). If VRAINIT is specified, the default is NOTSET. If VRAINIT is not specified, the default is SET. If you specify NOTSET or default to it, the control program places the address of the VRA plus the current length in the register before updating the VRA.

After updating the VRA, the control program updates the register to point to the next available space in the VRA. If you do not specify VRAREG, register 14 is the default.

,WORKREG=*reg*

specifies a work register. Each time you invoke the VRADATA macro instruction, the contents of this register are destroyed. If you do not specify WORKREG, register 15 is the default.

,TYPE= { **LEN,** **TEST** | **NOLEN,** **NOTEST** }

specifies whether (LEN) or not (NOLEN) you want the current length of the VRA stored in the VRALEN area and also specifies whether (TEST) or not (NOTEST) you want the VRA tested to see if it is full before adding the new entry. If you specify TEST, the current length of the VRA must already be in the VRACLEN area.

If you do not need to store the length or test to see if the new entry fits, specify NOLEN and NOTEST. These specifications considerably reduce the amount of code generated by the VRADATA macro instruction. If you do not specify TYPE, the value LEN, TEST is the default.

Notes:

1. *You must include the IHASDWA mapping macro instruction as a DSECT in your program if you accept the default for VRAINIT, VRACLEN, or VRAMLEN or if you specify VRAINIT=SDWAVRA. You must also place the address of the SDWA control block into the SDWAREG register (or default register 1) if you accept the default for any of these three keywords.*

2. *You must include the IHAVRA mapping macro as a DSECT in your program. If you include the IHASDWA mapping macro. IHAVRA is automatically included.*

3. *You can issue VRADATA more than once in a module, but you need to specify VRAINIT, VRACLEN, and VRAMLEN only once for a particular series of updates to the VRA.*

4. *If you specify a key but no data, the length field for the VRA entry is zero.*

Example 1

Operation: Initialize the SDWA control block to indicate that the VRA contains hexadecimal data, in key, length, data format. Also, move two pieces of data into the SDWAVRA, and indicate that no test of the length of the VRA is needed, (because the data fits in the VRA). The second request indicates that the length used is to be stored in the VRA current length field. The pieces of data are the IHAVRA mapping macro name and the contents of a control block.

```
VRADATA VRAINIT=SDWAVRA,KEY=VRACBM,DATA=MYCBNAME,          X
        TYPE=(NOLEN,NOTEST)
VRADATA KEY=VRACB,DATA=MYCB,TYPE=(LEN,NOTEST)
```

Example 2

Operation: Initialize a variable recording area that is not the SDWA. Move in a piece of data, specifying its length. (The piece of data is an ASID.)

```
VRADATA VRAINIT=LRBTUSR,VRACLEN=LRBTCLEN,                  X
        VRAMLEN=LBRTMLEN
VRADATA KEY=VRAAID,DATA=(REGA),LEN=ASIDLEN
```

VSMLIST - List Virtual Storage Map

The VSMLIST macro instruction provides information about the allocation of virtual storage. All addresses returned by the macro are 31-bit addresses. The information is returned in a work area that you specify. You must set bytes 0-3 of the work area to zero before the first invocation of this macro instruction for a specific request. The format of the work area is described in the "Virtual Storage Management" section in Volume 1.

This macro instruction can be used in cross memory mode. All addresses are associated with the current address space.

The following information can be requested:

- The ranges of virtual storage allocated to the SQA, by subpool, and the free space within those ranges
- The ranges of virtual storage allocated to the CSA, by subpool, and the free space within those ranges
- The ranges of CSA space that are unallocated
- The ranges of virtual storage allocated to the LSQA in the current address space, by subpool, and the free space within those ranges
- The ranges of virtual storage allocated to private area subpools, by TCB, and the free space within those ranges
- The ranges of private area that are unallocated
- On entry to this macro instruction, register 13 must contain the address of a 72-byte save area. VSMLIST preserves registers 2-13.

Except for the TCB, all input parameters to this macro instruction can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

The VSMLIST macro instruction is written as follows:

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

SP=SQA SP=CSA SP=LSQA SP=PVT SP=*sp list addr*	*sp list addr:* RX-type address or register (0), (2
,WKAREA=*(addr,length)*	*addr:* RX-type address or register (0), (2) *length:* symbol, decimal digit, or register (0), (2) - (12).
,TCB=*(tcb addr)* ,TCB=*(tcb addr*,ALL) ,TCB=(, ALL)	**Default:** TCB address in PSATOLD. *tcb addr:* RX-type address or regi **Note:** The TCB parameter is required only for SRB routines, if SP=PVT or SP=*sp list addr* and the list contains private area subpools.
,SPACE=ALLOC ,SPACE=FREE ,SPACE=UNALLOC	**Default:** SPACE=ALLOC **Note:** SPACE=UNALLOC can be specified only for SP=CSA or SP=PVT.
,LOC=ANY ,LOC=BELOW	**Default:** LOC=ANY
,REAL	
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	**Default:** LINKAGE=SYSTEM

The parameters are explained as follows:

SP=SQA
SP=CSA
SP=LSQA
SP=PVT
SP=*sp list addr*

specify the storage areas for which information is requested. The following subpools are listed for the specified storage areas:

- SQA: 226, 239, 245
- CSA: 227, 228, 231, 241
- LSQA: 255
- PVT: 0, 1-127, 229, 230, 236, 237, 251, 252

GETMAIN/FREEMAIN processing translates subpool numbers 233, 234, 235, 253, and 254, in the LSQA, to subpool number 255 and subpool numbers 240 and 250, in the private area, to subpool number 0. VSMLIST reports the translated subpool numbers, not the original subpool numbers. In addition, VSMLIST does not report invalid subpool numbers (subpool numbers greater than 255) or undefined subpool numbers (for example, 128).

If SP=*sp list addr* is specified, the user must supply the address of a subpool list. The first halfword of the list contains the number of entries in the list. Each of the following halfwords in the list contains a subpool number. If a valid subpool number appears more than once in the subpool list, it is reported only once.

,WKAREA=(*addr, length***)**
indicates the address and length of a user-supplied work area. VSM uses this work area to hold the parameter list, control information, and data that is to be returned to the caller. The work area should begin on a word boundary and be a minimum of 4K bytes in length.

You must set bytes 0-3 of this work area to zero before the first invocation of VSMLIST for a specific request. See "Virtual Storage Management" in Volume 1 for a description of the work area.

,TCB=(*tcb addr***)**
,TCB=(*tcb addr***,ALL)**
,TCB=(,ALL)
specify the TCB associated with the virtual storage allocated to the private area subpools. The TCB must be located in the currently addressable address space. If ALL is specified, the storage associated with the TCB and all of its subtasks is reported.

Notes:

1. *If ALL is specified and the TCB is high in the task structure (for example, the TCB for RCT), more than one region could be listed. The regions in the private area are the RCT region, the V=V region, and the V=R region (for V=R jobs).*

2. *The TCB resides in storage below 16 megabytes.*

,SPACE=ALLOC
,SPACE=FREE
,SPACE=UNALLOC
specify whether allocated, allocated and free, or unallocated storage is to be reported.

ALLOC indicates that the virtual addresses and lengths of blocks of storage allocated to the specific area are to be listed.

FREE indicates that in addition to the information supplied by ALLOC, the virtual addresses and lengths of free space within the allocated blocks are to be listed.

UNALLOC indicates that the virtual addresses and lengths of unallocated blocks of storage are to be listed. Both TCB and REAL are ignored when UNALLOC is specified.

Note: An allocated block of storage is a block that is a multiple of 4K in size and contains some storage that has been allocated via a GETMAIN macro instruction. The free storage is the storage within an allocated block that has not been allocated via a GETMAIN macro instruction. An unallocated block of storage is a block that is a multiple of 4K in size and contains no allocated storage.

,LOC=ANY
,LOC=BELOW
indicate which VSM queues are to be searched. If LOC=ANY is specified, all of the VSM queues are searched. If LOC=BELOW is specified, only those queues with virtual storage below 16 megabytes are searched.

,REAL
indicates that the high order bit of the address field of the allocated block descriptor is to be set. If the storage block was allocated using any LOC specification of GETMAIN but LOC=(,BELOW), the indicator is turned on; if the storage block was allocated using the LOC=(,BELOW) parameter of the GETMAIN macro instruction, the indicator is turned off. If REAL is not specified the indicator remains zero.

,LINKAGE=SYSTEM
,LINKAGE=BRANCH
indicate whether the VSMLIST routine uses a PC instruction (LINKAGE=SYSTEM) or branch entry (LINKAGE=BRANCH) for linkage and whether the VSMLIST routine provides serialization and recovery.

If LINKAGE=SYSTEM is specified, the VSMLIST routine provides linkage using a PC instruction and also provides recovery and serialization. The user cannot hold a lock higher than the local lock.

The caller's secondary ASID is preserved when a PC is issued; however, the caller cannot be in secondary addressing mode when issuing the macro instruction.

Note: Serialization is not provided across calls to VSMLIST.

If LINKAGE=BRANCH is specified, the VSMLIST routine uses branch entry for linkage and does not provide recovery or serialization. The user must hold the following locks for requests in specific areas.

- VSMFIX lock for SQA requests
- VSMFIX lock for CSA requests
- Local lock for LSQA requests
- Local lock for PVT requests

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

Hexadecimal Code	Meaning
0	The macro executed successfully and all the information has been placed in the data portion of the work area.
4	The macro executed successfully, but additional information has not been reported because there was not enough room in the data portion of the work area. To obtain the missing information, the user can continue to issue the macro, with the same options, until the return code in register 15 is 0.
8	An error occurred in scanning VSM control blocks. The information in the data area is valid, but incomplete. This return code is obtained only by users who specify LINKAGE=SYSTEM because it is detected during recovery.
C	The work area was too small or either invalid parameters or invalid control information was detected. This return code is obtained only by users who specify LINKAGE=BRANCH. Users who specify LINKAGE=SYSTEM will receive a C78 abend.

Note: Bytes 0-3 of the work area also contain the return code. Prior to the first invocation of the VSMLIST macro instruction, the user must set these bytes to zero. If the return code is 4 and the user wants to re-invoke the VSMLIST macro instruction, these bytes must not be changed.

Example 1

Operation: List the ranges of the allocated and free storage in the SQA. Specify the address of the VSM work area in register 2 and the length of the work area in register 3.

```
VSMLIST SP=SQA,SPACE=FREE,WKAREA=((2),(3))
```

Example 2

Operation: List the ranges of the allocated storage in the CSA. Specify the address of the work area in register 2 and the length of the work area in register 3. Provide branch entry linkage.

```
VSMLIST SP=CSA,SPACE=ALLOC,WKAREA=((2),(3)),LINKAGE=BRANCH
```

Example 3

Operation: List the ranges of unallocated storage in the private area. The variable X contains the address of the work area, which has a length of 4096 bytes.

```
VSMLIST SP=PVT,SPACE=UNALLOC,WKAREA=(X,4096)
```

Example 4

Operation: List the ranges of allocated storage, below 16 megabytes, in each of the subpools specified in the subpool list at location Y. The variable X contains the address of the work area, which has a length of 4096 bytes.

```
VSMLIST SP=Y,SPACE=ALLOC,WKAREA=(X,4096),LOC=BELOW
```

VSMLOC - Verify Virtual Storage Allocation

The VSMLOC macro instruction verifies that a given storage area has been allocated using the GETMAIN macro instruction. All addresses communicated between the caller and the VSMLOC routine must be 31-bit addresses. You can use VSMLOC in cross memory mode. All addresses are associated with the current address space.

The VSMLOC macro instruction is written as follows:

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

SQA CSA LSQA PVT CPOOLFIX CPOOLPAG CPOOLLCL	
,AREA=*(addr,length)*	*addr:* RX-type address or register (0) - (12 *length:* symbol, decimal digit or register (0), (2) - (12). Use only with SQA, CSA, LSQA, and PVT.
,AREA=*addr*	*addr:* RX-type address or register (0) - (12). Use only with CPOOLFIX, CPOOLPAG, and CPOOLLCL.
TCB=*addr*	*addr:* RX-type address or register (0) - (12). Can only be specified with PVT.
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	**Default:** LINKAGE=SYSTEM

The parameters are explained as follows:

SQA
CSA
LSQA
PVT

used to verify that storage for SQA, CSA, LSQA, or PVT (private area storage) has been allocated in the current address space.

,AREA=(*addr,length*)
indicates the start of the virtual storage area (*addr*) and the length of the virtual storage area (*length*) to be verified.

,AREA=(*addr*)
indicates the start of the virtual storage area (*addr*) to be verified.

CPOOLFIX
used to verify that storage for a global fixed cell pool has been allocated. Users who obtain their storage from subpools 226, 227, 228, 239, or 245 should specify this keyword.

CPOOLPAG
used to verify that storage for a global pageable cell pool has been allocated. Users who obtain their storage from subpools 231 and 241 should specify this keyword.

CPOOLLCL
used to verify that storage for a local cell pool has been allocated. Users who obtain their storage from subpools 0-127, 229, 230, 233-237, 240, 250-255 should specify this keyword.

TCB
indicates that VSMLOC is to place the address of the TCB associated with the verified storage in the register or storage area specified by the TCB parameter. If the return code from VSMLOC is not zero, the register or storage area specified by the TCB parameter is set to zero. The TCB parameter can only be specified with PVT.

,LINKAGE=SYSTEM
,LINKAGE=BRANCH
indicates the type of linkage that VSMLOC is to use and also indicates whether the VSMLOC routine is to provide recovery and serialization.

If LINKAGE=SYSTEM is specified, the VSMLOC routine uses a PC instruction for linkage and provides recovery and serialization. The following restrictions apply:

- If LSQA, PVT, or CPOOLLCL is specified, the user cannot hold a lock higher than the local lock.
- If CPOOLPAG is specified, the user cannot hold a lock higher than the VSMPAG lock.
- If CSA, SQA, or CPOOLFIX is specified, the user cannot hold a lock higher than the VSMFIX lock.

The caller's secondary ASID is preserved when a PC is issued; however, the caller cannot be in secondary addressing mode when issuing the macro.

If LINKAGE=BRANCH is specified, the VSMLOC routine uses branch entry linkage and does not provide recovery or serialization. The user must provide serialization by holding the following locks:

- The VSMFIX lock for CSA, SQA, and CPOOLFIX requests
- The VSMPAG lock for CPOOLPAG requests
- The local lock for LSQA, CPOOLLCL, and private area storage requests.

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

Hexadecimal Code	Meaning
0	The virtual storage specified has been allocated in the given storage area.
4	The virtual storage is not in the specified area or overlaps free space or different subpools in the given area.
8	An error occurred in processing VSM control blocks. This return code is obtained only by users who specify LINKAGE=SYSTEM because the error condition is detected during recovery.
12	The input is invalid. This return code is obtained only by users who specify LINKAGE=BRANCH. Users who specify LINKAGE=SYSTEM will receive a C78 abend.

Register 0 contains the following information:

Byte(s)	Meaning
0	Storage type indicator set by the macro (irrelevant to user, but can be non-zero)
1-2	Reserved
3	Subpool ID if the return code in register 15 is 0; 0 if the return code in register 15 is not 0

Note: Users should mask off bytes 0-2 before attempting to use the subpool ID returned in register 0.

Example 1

Operation: Verify that the virtual storage, starting at the address given in register 2 and having a length specified in register 3, has been allocated in the SQA.

```
VSMLOC SQA,AREA=((2),(3))
```

Example 2

Operation: Verify that the 8-bytes of virtual storage starting at X have been allocated in the CSA. Use a PC instruction for linkage and let VSMLOC provide recovery and serialization.

```
VSMLOC CSA,AREA=(X,8),LINKAGE=SYSTEM
```

Example 3

Operation: Verify that the 8-bytes of virtual storage starting at the address specified in register 2 have been allocated in the LSQA. Use branch entry for linkage.

```
VSMLOC LSQA,AREA=((2),8),LINKAGE=BRANCH
```

Example 4

Operation: Verify that the virtual storage, starting at X and having a length specified in register 3, has been allocated in private area storage. Use branch entry for linkage.

```
VSMLOC PVT,AREA=(X,(3)),LINKAGE=BRANCH
```

Example 5

Operation: Verify that the 100 bytes of virtual storage starting at the address specified in register 1 have been allocated in private area storage. The address of the TCB associated with the storage verified is returned in register 4.

```
VSMLOC PVT,AREA=((1),100),TCB=(4),LINKAGE=BRANCH
```

VSMREGN - Obtain Private Area Region Size

The VSMREGN macro instruction provides the virtual starting address and sizes of the private area regions associated with a given TCB in the current address space.

VSMREGN runs in the state and key of the caller. You can use VSMREGN in cross memory mode. All addresses communicated between VSMREGN and the caller are 31-bit addresses, associated with the current address space. If the TCB default is not used, the caller must hold the local lock.

Except for the TCB, all input parameters to this macro instruction can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

The VSMREGN macro instruction is written as follows:

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

WKAREA=*addr*	*addr:* RX-type address or register (0) - (12).
,TCB=*tcb addr*	**Default:** (except for SRB routines) TCB *tcb addr:* RX-type address or register (0), (2) - (12).

The parameters are written as follows:

WKAREA = *addr*
indicates the virtual address of a 16-byte work area, which is used by VSMREGN to return the requested information. The format of the work area is:

Bytes	Meaning
0-3	Virtual address of the region below 16 megabytes
4-7	Length of the region below 16 megabytes
8-11	Virtual address of the region above 16 megabytes
12-15	Length of the region above 16 megabytes

,TCB=*tcb addr*

indicates the virtual address of the TCB to be used to identify the region (the region control task (RCT) region, the V=V region, or the V=R region). SRB routines and routines whose currently addressable address space is not the home address space must specify the TCB operand. They cannot use the default value.

Note: The TCB resides in storage below 16 megabytes.

When control returns from the VSMREGN routine, register 15 contains the following return code:

Hexadecimal Code	Meaning
0	Successful completion

Example 1

Operation: Find the virtual address and length of the private area of the TCB whose address is in PSATOLD. Return the information in the work area whose address is given in register 2.

```
VSMREGN WKAREA=(2)
```

Example 2

Operation: Find the virtual address and length of the private area of the TCB specified in register 3. Return this information in the work area whose address is given in register 2.

```
VSMREGN WKAREA=(2),TCB=(3)
```

Example 3

Operation: Find the virtual address and length of the private area of the TCB whose address is X. Return this information in the work area whose address is given in register 2.

```
VSMREGN WKAREA=(2),TCB=X
```

Example 4

Operation: Find the virtual address and length of the private area of the TCB whose address is given in register 3. Return this information in the work area whose address is X.

```
VSMREGN WKAREA=X,TCB=(3)
```

WTL — Write To Log

The WTL macro instruction causes a message to be written to the system log. The message can include any character that can be used in a C-type (character) DC statement, and is assembled as a variable-length record. If the OPTION keyword is used by a non-authorized user, it is ignored.

The description of the WTL macro instruction follows. The WTL macro instruction is also described in *Supervisor Services and Macro Instructions* with the exception of the OPTION parameter. The use of the OPTION parameter is restricted to users who are authorized (APF-authorized, in system key 0-7, or in supervisor state).

Note: The exact format of the output of the WTL macro instruction varies depending on the job entry subsystem (JES2 or JES3) that is being used, the output class that is assigned to the log at system initialization, and whether DLOG is in effect for JES3. In JES3, system log entries are preceded by a 23-character prefix that includes a time stamp and routing information. If the combined prefix and message exceeds 126 characters, the log entry is truncated at the first blank or comma encountered when scanning backward from the 126th character of the combined prefix and message. See *Operations: JES3 Commands* for information about the format of the log entry when using JES3.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede WTL.
WTL	
b	One or more blanks must follow WTL.
'msg'	*msg:* Up to 126 characters if OPTION = NOPREFIX is specified. Up to 128 characters if OPTION = PREFIX is specified.
,OPTION = PREFIX ,OPTION = NOPREFIX	**Default:** OPTION = NOPREFIX

The parameters are explained as follows:

'msg'

specifies the message to be written to the system log. The message must be enclosed in apostrophes, which will not appear in the system log. See Figure 15 for a list of the printable EBCDIC characters passed to display devices or printers.

,OPTION=PREFIX
,OPTION=NOPREFIX

specifies whether the WTL text contains a prefix identifying the system log record. If PREFIX is specified, the text already contains a prefix. If NOPREFIX is specified or if this parameter is omitted, a 2-character prefix will be added by the control program. The OPTION keyword is ignored by any program running in the JES3 primary address space.

Note: If the msg text exceeds 126 characters, truncation occurs at the last embedded blank before the 126th character; when there are no embedded blanks, truncation occurs after the 126th character.

Hex Code	EBCDIC Character	Hex Code	EBCDIC Character	Hex Code	EBCDIC Character	Hex Code	EBCDIC Character
40	(space)	7B	#	99	r	D5	N
4A	¢	7C	@	A2	s	D6	O
4B	.	7D	'	A3	t	D7	P
4C	<	7E	=	A4	u	D8	Q
4D	(	7F	"	A5	v	D9	R
4E	+	81	a	A6	w	E2	S
4F	|	82	b	A7	x	E3	T
50	&	83	c	A8	y	E4	U
5A	!	84	d	A9	z	E5	V
5B	$	85	e	C1	A	E6	W
5C	*	86	f	C2	B	E7	X
5D	)	87	g	C3	C	E8	Y
5E	;	88	h	C4	D	E9	Z
5F	¬	89	i	C5	E	F0	0
60	-	91	j	C6	F	F1	1
61	/	92	k	C7	G	F2	2
6B	,	93	l	C8	H	F3	3
6C	%	94	m	C9	I	F4	4
6D	_	95	n	D1	J	F5	5
6E	>	96	o	D2	K	F6	6
6F	?	97	p	D3	L	F7	7
7A	:	98	q	D4	M	F8	8
						F9	9

Figure 15. Characters Printed or Displayed on an MCS Console

Notes:

1. *If the display service or printer is defined to JES3, the following characters are translated to blanks:*

   | ! ; ¬ : "

2. *The system recognizes the following hexadecimal representations of the U.S. national characters: @ as X'7C'; $ as X'5B'; and # as X'7B'. In countries other than the U.S., the U.S. national characters represented on terminal keyboards might generate a different hexadecimal representation and cause an error. For example, in some countries the $ character generates a X'4A'.*

Example 1

Operation: Write a message to the system log.

```
WTL    'THIS IS THE STANDARD FORMAT FOR THE WTL MACRO'
```

Example 2

Operation: Write a message to the system log specifying a prefix to identify the system log record.

```
WTL    'QL THIS FORMAT OF THE WTL USES THE OPTION KEYWORD',OPTION=PREFIX
```

WTL (List Form)

The list form of the WTL macro instruction is used to construct a control program parameter list. The message parameter must be provided in the list form of the macro instruction.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede WTL.
WTL	
b	One or more blanks must follow WTL.
'msg'	*msg:* Up to 126 characters.
,MF=L	

The *'msg'* parameter is explained under the standard form of the WTL macro instruction. The OPTION keyword is not permitted on the list form of the WTL macro. A description of the MF parameter follows:

,MF=L
specifies the list form of the WTL macro instruction.

Note: If msg text exceeds 126 characters, truncation occurs at the last embedded blank before the 126th character; when there are no embedded blanks, truncation occurs after the 126th character.

Example 1

Operation: Build a parameter list for a message to be written to the system log.

```
LOGMSG    WTL    'FUNCTION XXX COMPLETE',MF=L
```

WTL (Execute Form)

The execute form of the WTL macro instruction uses a remote control program parameter list. The parameter list can be generated by the list form of WTL. You cannot modify the message in the execute form.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede WTL.
WTL	
b	One or more blanks must follow WTL.
MF=(E,*ctrl addr*)	*ctrl addr:* RX-type address, or register (1) or (2) - (12).
,OPTION=PREFIX ,OPTION=NOPREFIX	**Default:** OPTION=NOPREFIX

The OPTION parameter is explained under the standard form of the WTL macro instruction; this parameter is explained as follows:

MF=(E,*ctrl addr*)
specifies the execute form of the WTL macro instruction. This form uses a remote control program parameter list.

Example 1

Operation: Write a message constructed in the list form of WTL.

```
WTL    MF=(E,LOGMSG)
```

WTO - Write to Operator

The WTO macro instruction causes a message to be written to one or more operator consoles.

An authorized user (supervisor state with protection key 0-7) can issue a multiple line WTO message of up to 255 lines with one WTO macro instruction. If you are coding a multiple line message, you must ensure that the left-most three bytes of register 0 are set correctly. For the first request (of up to 255 lines), these three bytes must be zero. For subsequent requests, the first three bytes of register 0 may contain the message identifier that the WTO service routine returns in register 1 after the first request. The CONNECT keyword provides another way to connect WTO messages.

Use of the MSGTYP and MCSFLAG parameters should only be attempted by system programmers familiar with MCS, because using these parameters improperly might interfere with the message routing scheme.

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

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

'msg' *('text')* *('text',line type)*	*msg:* Up to 125 characters. *text:* Up to 125 characters. The permissible *line types*, text lengths, and maximum numbers are shown below:

line type	text	maximum number
C	34 char	1 C type
L	70 char	2 L type
D	70 char	255 D type (authorized programs) 10 D type (unauthorized programs)
DE	70 char or	1 DE type
E	None	1 E type

The maximum total number of line types that can be coded in one instruction is 255.

,ROUTCDE = *(routing code)*	*routing code:* decimal digit from 1 to 128. The *routing code* is one or more codes, separated by commas, or a hyphen to indicate a range.
,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 = *(flag name)*	*flag name:* any combination of the following, separated by commas:

REG0	HRDCPY	CMD
RESP	QREG0	BUSYEXIT
REPLY	NOTIME	
BRDCST	NOCPY	

,CONNECT = *addr*	*addr:* RX-type address or register
,CONSID = *addr*	*addr:* RX-type address or register
,KEY = *addr*	*addr:* RX-type address or register
,TOKEN = *addr*	*addr:* RX-type address or register
,JOBID = *addr*	*addr:* RX-type address or register
,JOBNAME = *addr*	*addr:* RX-type address or register
,SYSNAME = *addr*	*addr:* RX-type address or register
,WQEBLK = *addr*	*addr:* RX-type address or register
,PRTY = *addr*	*addr:* RX-type address or register
,SUBSMOD = YES ,SUBSMOD = NO	**Default:** YES

The parameters are explained as follows:

'msg'
('text')
('text',line type)
specifies the message or multiple-line message to be written to one or more operator consoles.

The first format is used to write a single-line message to the operator. In the format, the message must be enclosed in apostrophes, which do not appear on the console. It can include any character that can be used in a character (C-type) DC instruction. When a program issues a WTO macro instruction, the control program translates the text; only standard printable EBCDIC characters, shown in Volume 1 under the topic "Writing Operator Messages." are passed to the display devices. All other characters are replaced by blanks. If the terminal does not have dual-case capability, it prints lowercase characters as uppercase. The message is assembled as a variable-length record.

The second and third formats are used to write a multiple-line message to the operator. For a problem program the message can be up to ten lines long; the system truncates the message at the end of the tenth line. The ten-line limit does not include the control line (message IEE932I), as explained under line type C below. The message can be up to 255 lines long for an authorized program.

Note: If the second format is coded without repetition, for example, *('text')*, the message appears as a single-line message.

The text is one line of the multiple-line message. A line consists of a character string enclosed in apostrophes (which do not appear on the operator console). Any character valid in a C-type DC instruction can be coded. The maximum number of characters depends on which line type is specified.

The line type defines the type of information contained in the 'text' field of each line of the message:

C

indicates that the 'text' parameter is the text to be contained in the control line of the message. The control line normally contains a message title. C may only be coded for the first line of a multiple-line message. If this parameter is omitted and descriptor code 9 is coded, the system generates a control line (message IEE932I) containing only a message identification number. The control line remains static during framing operations on a display console (provided that the message is displayed in an out-of-line display area). Control lines are optional.

L

indicates that the 'text' parameter is a label line. Label lines contain message heading information; they remain static during framing operations on a display console (provided that the message is displayed in an out-of-line display area). Label lines are optional. If coded, lines must either immediately follow the control line or another label line or be the first line of the multiple-line message if there is no control line. Only two label lines may be coded per message. See the topic "Embedding Label Lines in a Multiline Message" in Volume 1 for additional information concerning how to include multiple label lines (more than two) within a message.

D

indicates that the 'text' parameter contains the information to be conveyed to the operator by the multiple-line message. During framing operations on a display console, the data lines are paged.

DE

indicates that the 'text' parameter contains the last line of information to be passed to the operator.

E

indicates that the previous line of text was the last line of text to be passed to the operator. The 'text' parameter, if any, coded with a line type of E is ignored.

,ROUTCDE = *routing code*

specifies the routing code(s) to be assigned to the message.

The routing codes are:

1 Master console action
2 Master console information
3 Tape pool
4 Direct access pool
5 Tape library
6 Disk library
7 Unit record pool
8 Teleprocessing control
9 System security
10 System error/maintenance
11 Programmer information
12 Emulators

Note: The values for this operand can be from 1 to 128. Routing codes 13 through 20 are reserved for customer use. Routing codes 29 through 41 are reserved, and are ignored if specified. Routing codes 42 through 128 are available to authorized programs only. Routing codes 1, 2, 3, 4, 7, 8, 10 and 42 cause hard copy of the message when display consoles are used or more than one console is active. All other routing codes may go to hard copy as an initialization option or as a result of a VARY HARDCPY command.

,DESC = *(desc code)*

specifies the message descriptor code(s) to be assigned to the message. Descriptor codes 1 through 6 and descriptor code 11 are mutually exclusive. Codes 7 through 10 can be assigned in combination with any other code.

The descriptor codes are:

1 System failure
2 Immediate action required
3 Eventual action required
4 System Status
5 Immediate command response
6 Job status
7 Application program/processor; message is to be deleted when issuing task is terminated
8 Out-of-line message
9 Operator request
10 Dynamic status displays
11 Critical eventual action requested
12-16 Reserved for future use

All WTO messages with descriptor codes 1, 2, or 11 are action messages that have an @ or * sign displayed before the first character of the message. This indicates a need for operator action. On operator consoles that support color, descriptor codes determine the color in which a message should be displayed. The colors used for different descriptor codes are described in *Operations: System Commands.*

,AREAID = *id char*

specifies a display area of the console screen on which a multiple-line message is to be written. This parameter is meaningful only for out-of-line (descriptor code 8 and 9) MLWTO messages that are to be sent to display 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 an authorized (supervisor state, key 0-7, or APF-authorized) user to use this parameter. Otherwise, Z is used. Also, if this parameter specifies an area, the area could be overlaid by a currently running dynamic display. Support for queuing messages with descriptor code 8 is by console id only.

,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 that issued the MONITOR SESS, MONITOR JOBNAMES, or MONITOR STATUS command, respectively. When the message type is identified by the operating system, the message is routed only to those consoles that 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 = *(flag name)*

specifies that the macro expansion should set bits in the MCSFLAG field as indicated by each name coded.

The flag names and meanings are shown in Figure 16.

Flag Name	Meaning
REG0	Queue the message to the console whose source ID is passed in register 0.
RESP	The WTO is an immediate command response.
REPLY	This WTO is a reply to a WTOR.
BRDCST	Broadcast the message to all active consoles.
HRDCPY	Queue the message for hard copy only.
QREG0	Queue the message unconditionally to the console whose source ID is passed in Register 0.
NOTIME	Do not append time to the message.
NOCPY	If the WTO or WTOR macro instruction is issued by a program in the supervisor state, do not queue the message for hard copy. Otherwise, this parameter is ignored.
CMD	The WTO is a recording of a system command issued for hardcopy log purposes.
BUSYEXIT	If there are no message or console buffers for either MCS or JES3, or there is a JES3 WTO staging area excess, the WTO is terminated with a x'20' return code. If BUSYEXIT is not specified, the WTO will go into a wait state if WTO buffers are not available.

Figure 16. MCSFLAG Fields (WTO)

CONNECT = *addr*
specifies a field containing the 4-byte message number of the previous WTO that this WTO is to be connected to. This message number is obtained as an output parameter (returned in register 1) from the previous WTO. This keyword is mutually exclusive with the CONSID and SYSNAME keywords, and it is valid only for continuation of multiple-line messages. When this keyword is specified in the list form, it must be coded as CONNECT= with nothing after the =.

Note: You can still use the register interface, mentioned at the beginning of the WTO macro description, to connect WTO messages. If you specify both, however, the system uses the CONNECT keyword. It is recommended that new users use the CONNECT keyword.

CONSID = *addr*
specifies a field containing the 4-byte id of the console to receive a message. This id is used in place of a console id in register 0. This keyword is mutually exclusive with the CONNECT keyword. Note that you can still specify the console ID via the register zero interface. If you use both the CONSID keyword and the register zero interface to specify the console ID, the system uses the console ID specified by CONSID. When this keyword is specified in the list form, it must be coded as CONSID= with nothing after the =.

KEY = *addr*
specifies a field containing an 8-byte key to be associated with this message. The key should be EBCDIC if used with the D R command for retrieval purposes, but it must not be '*'. If a register is used, it contains the address of the key. When this keyword is specified in the list form, it must be coded as KEY= with nothing after the =.

TOKEN = *addr*
specifies a field containing a 4-byte token to be associated with this message. This field is used to identify a group of messages that can be deleted by a DOM macro that includes TOKEN. The token should be unique within an address space. When this keyword is specified in the list form, it must be coded as TOKEN= with nothing after the =.

WQEBLK = *addr*
specifies a field containing a WQE control block to be used as input for the WTO. If a register is used, the register contains the address of the WQE control block to be used as input for the WTO. This keyword is mutually exclusive with all other keywords and reserved for authorized users. It is valid only on the standard form of the macro. When this keyword is specified in the list form, it must be coded as WQEBLK= with nothing after the =.

JOBNAME = *addr*
specifies a field containing the 8-byte job name with which the JES identifies the issuer of this WTO. If the value of the job name is less than 8 characters, it must be left-justified and padded with blanks. If a register is used, the register contains the address of the 8-byte job name. This keyword is reserved for authorized users. When this keyword is specified in the list form, it must be coded as JOBNAME= with nothing after the =.

JOBID = *addr*
specifies a field containing the 8-byte EBCDIC job identifier (number) with which JES identifies the issuer of this WTO. If the value of the job id is less than 8 characters, it should be right-justified although this is not absolutely necessary. If a register is used, the register contains the address of the 8-byte job id. When this keyword is specified in the list form, it must be coded as JOBID= with nothing after the =. This keyword is reserved for authorized users.

SUBSMOD=YES
SUBSMOD=NO
specifies whether the characteristics of the message may be modified by the primary subsystem. YES, which is the default value, indicates that the message may be modified by the primary subsystem. NO indicates that the message may not be modified by the primary subsystem. This keyword is reserved for authorized users.

PRTY=*addr*
specifies a field containing the two-byte priority to be assigned to this message. If used, the register contains a 2-byte priority. When this keyword is specified in the list form, it must be coded as PRTY= with nothing after the =. This keyword is reserved for authorized users.

SYSNAME=*addr*
specifies a field containing an 8-byte system name to be associated with this message. If a register is used, the register contains the address of the 8-byte system name. This keyword is mutually exclusive with CONNECT and it is reserved for authorized users. When specified in the list form, it must be coded as SYSNAME= with nothing after the =.

When control is returned, general register 1 contains the identification number (24 bits and right-justified) assigned to the message. If you are using the CONNECT keyword to connect WTO messages, store this value in the four-byte CONNECT field and set register 1 to zero.

Return codes from execution of a WTO are as follows:

Hexadecimal Code	Meaning
00	No errors encountered.
04	Number of lines passed was 0; request is ignored. Number of lines passed was greater than 10; only 10 lines are processed. Message text length for a line was less than 1; all lines up to error line are processed.
08	Connecting message ID passed in register 0 does not match any on queue. Request is ignored.
0C	Invalid line type. An end has been forced at the point of the error except if the first line is an E line, in which case the request is ignored.
20	WTO processing has been terminated since it would have caused a wait state, and BUSYEXIT was specified.
30	Required resource for routing code 11 was not available. Request is ignored for routing code 11; if any other routing code is specified, the request is processed.

Notes:

1. *If the list and execute forms of the WTO macro are in separate modules, both routines must be assembled or compiled with the same level of WTO.*

2. *If the execute form of the macro specifies CONNECT, CONSID, KEY, TOKEN, JOBID, JOBNAME, SYSNAME, or PRTY, then the list form, to ensure that the parameter list is generated correctly, must specify the same keyword(s) without data. For example:*

   ```
   WTO 'text',SYSNAME=,CONSID=,MF=L
   ```

   If data is specified, the system issues an MNOTE and ignores the data.

3. *For any WTO keywords that allow a register specification, the value must be right-justified in the register.*

4. *Use caution when coding a program that uses any of the WTO keywords reserved for authorized users.*

Example 1

Operation: Issue a WTO that describes a situation that must be resolved immediately. The message must appear on the master console and the system maintenance console as an immediate action message.

```
WTO  'THIS IS AN IMMEDIATE ACTION MESSAGE',ROUTCDE=(1,10),DESC=(2)
```

Example 2

Operation: Issue a message that is the response to a command. The message must appear only on the console that entered the command. Prior to issuing the WTO, the issuing console's identifier must be placed in register 0.

```
WTO  'THIS IS A COMMAND RESPONSE',DESC=(5),MCSFLAG=(REG0,RESP)
```

WTO (List Form)

The list form of the WTO macro instruction is used to construct a control program parameter list.

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

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

'msg' *('text')* *('text',line type)*	*msg:* Up to 125 characters. *text:* Up to 125 characters. The permissible *line types*, text lengths, and maximum numbers are shown below: line type / text / maximum number C / 34 char / 1 C type L / 70 char / 2 L type D / 70 char / 255 D type (authorized programs); 10 D type (unauthorized programs) DE / 70 char or / 1 DE type E / None / 1 E type The maximum total number of line types that can be coded in one instruction is 255.
,ROUTCDE=*(routing code)*	*routing code:* decimal digit from 1 to 128. The *routing 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=*(flag name)*	*flag name:* any combination of the following, separated by commas: REG0, HRDCPY, CMD RESP, QREG0, BUSYEXIT REPLY, NOTIME BRDCST, NOCPY
,MF=L	
,CONNECT=	
,CONSID=	
,KEY=	
,TOKEN=	
,JOBID=	
,JOBNAME=	
,SYSNAME=	
,PRTY=	
,SUBSMOD=YES ,SUBSMOD=NO	

The parameters are explained under the standard form of the WTO macro instruction, with the following exception:

,MF=L
specifies the list form of the WTO macro instruction.

Notes:

1. *If the list and execute forms of the WTO macro are in separate modules, both routines must be assembled or compiled with the same level of WTO.*

2. *If the execute form of the macro specifies CONNECT, CONSID, KEY, TOKEN, JOBID, JOBNAME, SYSNAME, or PRTY, then the list form, to ensure that the parameter list is generated correctly, must specify the same keyword(s) without data. For example:*

   ```
   WTO 'text',SYSNAME=,CONSID=,MF=L
   ```

   If data is specified, the system issues an MNOTE and ignores the data.

3. *For any WTO keywords that allow a register specification, the value must be right-justified in the register.*

WTO (Execute Form)

The execute form of the WTO macro instruction uses a remote control program parameter list. The parameter list can be generated by the list form of WTO. The message text cannot be modified in the execute form of the macro instruction.

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

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

,CONNECT=*addr*	*addr:* RX-type address or register
,CONSID=*addr*	*addr:* RX-type address or register
,KEY=*addr*	*addr:* RX-type address or register
,TOKEN=*addr*	*addr:* RX-type address or register
,JOBID=*addr*	*addr:* RX-type address or register
,JOBNAME=*addr*	*addr:* RX-type address or register
,SYSNAME=*addr*	*addr:* RX-type address or register
,WQEBLK=*addr*	*addr:* RX-type address or register
,PRTY=*addr*	*addr:* RX-type address or register
MF=*(E,ctrl addr)*	*ctrl addr:* RX-type address, or register (1) or (2) - (12).

This parameter is explained under the standard form of the WTO macro instruction, with the following exceptions:

MF=(E, *ctrl addr)*
specifies the execute form of the WTO macro instruction using a remote control program parameter list.

Notes:

1. *If the list and execute forms of the WTO macro are in separate modules, both routines must be assembled or compiled with the same level of WTO.*

2. *If the execute form of the macro specifies CONNECT, CONSID, KEY, TOKEN, JOBID, JOBNAME, SYSNAME, or PRTY, then the list form, to ensure that the parameter list is generated correctly, must specify the same keyword(s) without data. For example:*

   ```
   WTO 'text',SYSNAME=,CONSID=,MF=L
   ```

   If data is specified, the system issues an MNOTE and ignores the data.

3. *For any WTO keywords that allow a register specification, the value must be right-justified in the register.*

Example

Operation: Write a message with a pre-built parameter list pointed to by register 1.

```
WTO     MF=(E,(1))
```

WTOR - Write to Operator with Reply

The WTOR macro instruction causes a message requiring a reply to be written to one or more operator consoles and the hardcopy log. The macro instruction also provides the information required by the control program to return the reply to the issuing program.

This macro can be assembled compatibly between MVS/XA and MVS/370 through the use of the SPLEVEL macro instruction. Default processing will result in an expansion of the macro that operates only with MVS/XA. See the topic "Selecting the Macro Level" for additional information. If you are executing in 31-bit addressing mode, you must use the MVS/XA version of this macro instruction.

Use of the MSGTYP and MCSFLAG parameters should only be attempted by system programmers familiar with MCS, because using these parameters improperly could impede the entire message routing scheme.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede WTOR.
WTOR	
b	One or more blanks must follow WTOR.
msg	*msg:* up to 122 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 = *(routing code)*	*routing code:* decimal digit from 1 to 128. The *routing 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 combination of the following, separated by commas: N Y SESS JOBNAMES STATUS
,MCSFLAG = *(flag name)*	*flag name:* any combination of the following, separated by commas: REG0 HRDCPY CMD RESP QREG0 REPLY NOTIME BRDCST NOCPY
,CONSID = *addr*	*addr:* RX-type address or register
,KEY = *addr*	*addr:* RX-type address or register
,TOKEN = *addr*	*addr:* RX-type address or register
,JOBID = *addr*	*addr:* RX-type address or register
,JOBNAME = *addr*	*addr:* RX-type address or register
,SYSNAME = *addr*	*addr:* RX-type address or register
,WQEBLK = *addr*	*addr:* RX-type address or register

The parameters are explained as follows:

'msg'

specifies the message to be written to the operator's console. The message must be enclosed in apostrophes, which do not appear on the console. It can include any character that can be used in a character (C-type) DC instruction. When a program issues a WTOR macro instruction, the control program translates the text; only standard printable EBCDIC characters are passed to the display devices. See the figure "EBCDIC Characters Printed or Displayed on an MCS Console" in Volume 1 for a list of the printable EBCDIC characters. All other characters are replaced by blanks. If the terminal does not have dual-case capability, it prints lowercase characters as uppercase. The message is assembled as a variable-length record.

Note: All WTOR messages are action messages. An indicator appears before the first character of an action message to indicate a need for operator action.

,reply addr

specifies the address in virtual storage of the area into which the control program is to place the reply. The reply is left-justified at this address.

,reply length

specifies the length, in bytes, of the reply message.

,ecb addr

specifies the address of the event control block (ECB) to be used by the control program to indicate the completion of the reply and the id of the replying console. After the control program receives the reply, the ECB appears as follows:

Offset	Length(bytes)	Contents
0	1	Completion code
1	2	Reserved
3	1	Console id in hexadecimal

,ROUTCDE=*routing code*

specifies the routing code(s) to be assigned to the message.

The routing codes are:

1 Master console action
2 Master console information
3 Tape pool
4 Direct access pool
5 Tape library
6 Disk library
7 Unit record pool
8 Teleprocessing control
9 System security
10 System error/maintenance
11 Programmer information
12 Emulators

Note: The values for this operand can be from 1 to 128. Routing codes 13 through 20 are reserved for customer use. Routing codes 29 through 41 are reserved, and are ignored if specified. Routing codes 42 through 128 are available to authorized programs only. Routing codes 1, 2, 3, 4, 7, 8, 10, and 42 cause hard copy of the message when display consoles are used or more than one console is active. All other routing codes may go to hard copy as an initialization option or as a result of a VARY HARDCPY command.

,DESC = *(desc code)*

specifies the message descriptor code(s) to be assigned to the message. Descriptor codes 1 through 6 and descriptor code 11 are mutually exclusive. Codes 7 through 10 can be assigned in combination with any other code.

The descriptor codes are:

1 System failure
2 Immediate action required
3 Eventual action required
4 System Status
5 Immediate command response
6 Job status
7 Application program/processor; message is to be deleted when issuing task is terminated
8 Out-of-line message
9 Operator request
10 Dynamic status displays
11 Critical eventual action requested
12-16 Reserved for future use

All WTOR messages with descriptor codes 1, 2, or 11 are action messages that have an @ or * character displayed before the first character of the message. This character indicates a need for operator action. On operator consoles that support color, descriptor codes determine the color in which a message should be displayed. The colors used for different descriptor codes are described in *Operations: System Commands.*

,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 that issued the MONITOR SESS, MONITOR JOBNAMES, or MONITOR STATUS command, respectively. When the message type is identified by the operating system, the message is routed only to those consoles that requested the information.

For Y or N, the message type specifies whether flags are to be set in the WTOR 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 = *(flag name)*

specifies that the macro expansion should set bits in the MCSFLAG field as indicated by each name coded.

The flag names and meanings are shown in Figure 17.

Flag Name	Meaning
REG0	Queue the message to the console whose source ID is passed in register 0.
RESP	The WTOR is an immediate command response.
REPLY	This is a reply to a WTOR.
BRDCST	Broadcast the message to all active consoles.
HRDCPY	Queue the message for hard copy only.
QREG0	Queue the message unconditionally to the console whose source ID is passed in Register 0.
NOTIME	Do not append time to the message.
NOCPY	If the WTOR macro instruction is issued by a program in the supervisor state, do not queue the message for hard copy. Otherwise, this parameter is ignored.
CMD	The WTOR is a recording of a system command issued for hardcopy log purposes.

Figure 17. MCSFLAG Fields (WTOR)

CONSID = *addr*
specifies a field containing the 4-byte id of the console to receive a message. This id is used in place of a console id in register 0. Note that you can still specify the console ID via the register zero interface. If you use both the CONSID keyword and the register zero interface to specify the console ID, the system uses the console ID specified by CONSID. When this keyword is specified in the list form, it must be coded as CONSID= with nothing after the =.

KEY = *addr*
specifies a field containing an 8-byte key to be associated with this message. The key should be EBCDIC if used with the D R command for retrieval purposes, but it must not be '*'. If a register is used, it contains the address of the key. When this keyword is specified in the list form, it must be coded as KEY= with nothing after the =.

TOKEN = *addr*
specifies a field containing a 4-byte token to be associated with this message. This field is used to identify a group of messages that can be deleted by a DOM macro that includes TOKEN. The token should be unique within an address space. When this keyword is specified in the list form, it must be coded as TOKEN= with nothing after the =.

WQEBLK = *addr*
specifies a field containing a WQE control block to be used as input for the WTOR. If a register is used, the register contains the address of the WQE control block to be used as input for the WTOR. This keyword is mutually exclusive with all other keywords and reserved for authorized users. It is valid only on the standard form of the macro. When this keyword is specified in the list form, it must be coded as WQEBLK= with nothing after the =.

JOBNAME = *addr*
specifies a field containing the 8-byte job name with which the JES identifies the issuer of this WTOR. If the value of the job name is less than 8 characters, it must be left-justified and padded with blanks. If a register is used, the register contains the address of the 8-byte job name. This keyword is reserved for authorized users. When this keyword is specified in the list form, it must be coded as JOBNAME= with nothing after the =.

JOBID = *addr*

specifies a field containing the 8-byte EBCDIC job identifier (number) with which JES identifies the issuer of this WTOR. If the value of the job id is less than 8 characters, it should be right-justified although this is not absolutely necessary. If a register is used, the register contains the address of the 8-byte job id. When this keyword is specified in the list form, it must be coded as JOBID= with nothing after the =. This keyword is reserved for authorized users.

SYSNAME = *addr*

specifies a field containing an 8-byte system name to be associated with this message. If a register is used, the register contains the address of the 8-byte system name. This keyword is reserved for authorized users. When specified in the list form, it must be coded as SYSNAME= with nothing after the =.

When control is returned, general register 1 contains the identification number (4 bytes or 32 bits and right-justified) assigned to the message.

Return codes from execution of a WTOR are as follows:

Hexadecimal Code	Meaning
00	No errors encountered.
04	Number of lines passed was 0; request is ignored. Number of lines passed was greater than 10; only 10 lines are processed. Message text length for a line was less than 1; all lines up to error line are processed.
08	Connecting message ID passed in register 0 does not match any on queue. Request is ignored.
0C	Invalid line type. An end has been forced at the point of the error except if the first line is an E line, in which case the request is ignored.
30	Required resource for routing code 11 was not available. Request is ignored for routing code 11; if any other routing code is specified, the request is processed.

Notes:

1. *If the list and execute forms of the WTOR macro are in separate modules, both routines must be assembled or compiled with the same level of WTOR.*

2. *For any WTOR keywords that allow a register specification, the value must be right-justified in the register.*

3. *Use caution when coding a program that uses any of the WTOR keywords reserved for authorized users.*

Example 1

Operation: Issue a WTOR that describes a situation that must be resolved immediately. The message must appear on the master console and the system maintenance console as an immediate action message.

```
WTOR  'THIS IS AN IMMEDIATE ACTION MESSAGE',ROUTCDE=(1,10),DESC=(2)
```

Example 2

Operation: Issue a message that is the response to a command. The message must appear only on the console that entered the command. Prior to issuing the WTOR, the issuing console's identifier must be placed in register 0.

```
WTOR  'THIS IS A COMMAND RESPONSE',DESC=(5),MCSFLAG=(REG0,RESP)
```

WTOR (List Form)

The list form of the WTOR macro instruction is used to construct a control program parameter list.

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

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede WTOR.
WTOR	
b	One or more blanks must follow WTOR.
msg	*msg:* up to 122 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).
,MF=L	
,ROUTCDE=*(routing code)*	*routing code:* decimal digit from 1 to 128. The *routing 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 combination of the following, separated by commas: N Y SESS JOBNAMES STATUS
,MCSFLAG=*(flag name)*	*flag name:* any combination of the following, separated by commas: REG0 HRDCPY CMD RESP QREG0 REPLY NOTIME BRDCST NOCPY
,CONSID=	
,KEY=	
,TOKEN=	
,JOBID=	
,JOBNAME=	
,SYSNAME=	

The parameters are explained under the standard form of the WTOR macro instruction, with the following exception:

,MF = L
specifies the list form of the WTOR macro instruction.

Notes:

1. *If the list and execute forms of the WTOR macro are in separate modules, both routines must be assembled or compiled with the same level of WTOR.*

2. *If the execute form of the macro specifies CONNECT, CONSID, KEY, TOKEN, JOBID, JOBNAME, SYSNAME, DOMID, or PRTY, then the list form, to ensure that the parameter list is generated correctly, must specify the same keyword(s) without data. For example:*

   ```
   WTO 'text',SYSNAME=,CONSID=,MF=L
   ```

   If data is specified, the system issues an MNOTE and ignores the data.

3. *For any WTOR keywords that allow a register specification, the value must be right-justified in the register.*

WTOR (Execute Form)

The execute form of the WTOR macro instruction uses a remote control program parameter list. The parameter list can be generated by the list form of WTOR. The message cannot be modified in the execute form of the macro instruction.

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

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

,CONSID=*addr*	*addr:* RX-type address or register
,KEY=*addr*	*addr:* RX-type address or register
,TOKEN=*addr*	*addr:* RX-type address or register
,JOBID=*addr*	*addr:* RX-type address or register
,JOBNAME=*addr*	*addr:* RX-type address or register
,SYSNAME=*addr*	*addr:* RX-type address or register
MF=*(E,ctrl addr)* *addr,EXTENDED)*	*ctrl addr:* RX-type address, or register (1) or (2) - (12). MF=*(E,ctrl*

This parameter is explained under the standard form of the WTO macro instruction, with the following exception:

MF=(E, *ctrl addr)* **MF=(E,** *ctrl addr,EXTENDED)*
specifies the execute form of the WTOR macro instruction using a remote control program parameter list. EXTENDED must be specified with any of the keywords listed in note 2.

Notes:

1. *If the list and execute forms of the WTOR macro are in separate modules, both routines must be assembled or compiled with the same level of WTOR.*

2. *If the execute form of the macro specifies CONSID, KEY, TOKEN, JOBID, JOBNAME, or SYSNAME then the list form, to ensure that the parameter list is generated correctly, must specify the same keyword(s) without data. For example:*

   ```
   WTOR 'text',SYSNAME=,CONSID=,MF=L
   ```

   If data is specified, the system issues an MNOTE and ignores the data.

3. *For any WTOR keywords that allow a register specification, the value must be right-justified in the register.*

Example

Operation: Write a message with a pre-built parameter list pointed to by register 1.

```
WTOR      MF=(E,(1))
```

Index

D

E

F

G

H

I

Q

R

S

T

U

V

W

MVS/Extended Architecture System Programming Library: System Macros and Facilities Volume 2

GC28-1151-4 S370-36

Printed in U.S.A.

MVS/Extended Architecture
System Programming
Library: System Macros
and Facilities
Volume 2

READER'S
COMMENT
FORM

GC28-1151-4

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

_____

_____

_____

_____

What is your occupation? _____

How do you use this publication? _____

Number of latest Newsletter associated with this publication: _____

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 or you may mail directly to the address in the Edition Notice on the back of the title page.)

MVS/Extended Architecture System Programming Library: System Macros and Facilities Volume 2

GC28-1151-4 S370-36

Reader's Comment Form

Cut or Fold Along Line

Fold and tape Please Do Not Staple Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department D58, Building 921-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape Please Do Not Staple Fold and tape

Printed in U.S.A.

IBM®

MVS/Extended Architecture
System Programming
Library: System Macros
and Facilities
Volume 2

READER'S
COMMENT
FORM

GC28-1151-4

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

_____

_____

_____

_____

What is your occupation? _____

How do you use this publication? _____

Number of latest Newsletter associated with this publication: _____

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 or you may mail directly to the address in the Edition Notice on the back of the title page.)

MVS/Extended Architecture System Programming Library: System Macros and Facilities Volume 2

GC28-1151-4 S370-36

Reader's Comment Form

Cut or Fold Along Line

Fold and tape Please Do Not Staple Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department D58, Building 921-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape Please Do Not Staple Fold and tape

Printed in U.S.A.

IBM®

GC28-1151-04