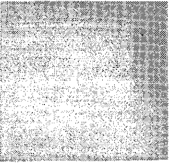
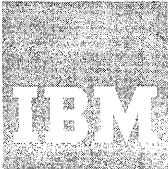
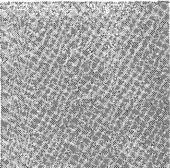
File No. S360-36
GC28-6646-7

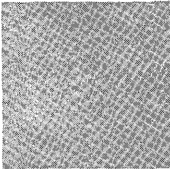




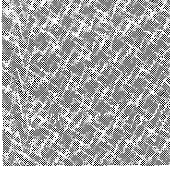
Systems Reference Library



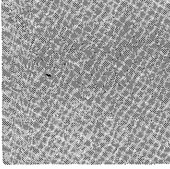
OS Release 21



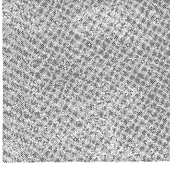
IBM System/360 Operating System Supervisor Services and Macro Instructions



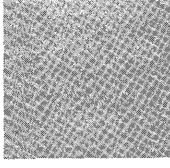
The title of this manual was formerly IBM System/360 Operating Supervisor Services. The descriptions of the supervisor macro instructions formerly found in IBM System/360 Operating System Supervisor and Data Management Macro Instructions, GC28-6647 have been added.

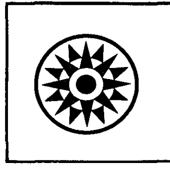


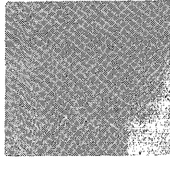
This manual describes how to use the services of the supervisor, the macro instructions used to request these services, and the linkage conventions used by the control program to provide these services. Included in the services of the supervisor are program management, task creation and management, and main-storage management.



Intended mainly for the programmer coding in assembler language, this book is a guide to using the macro instructions described. This book does not discuss macro instructions used for graphics, teleprocessing, optical readers, optical reader-sorters, or magnetic character readers. These macro instructions are discussed in separate publications that are listed in the IBM System/360 Bibliography, GA22-6822.







Eighth Edition (September, 1974)

This edition is a publisher's revision of GC28-6646-6, and incorporates TNL GN27-1419. This edition applies to OS Release 21.7

This publication now contains the descriptions of the supervisor macro instructions formerly contained in Supervisor and Data Management Macro Instructions, GC28-6647. In addition, technical changes and clarifications have been made throughout the book.

The information in the book changes from time to time. Before using this manual with IBM systems, consult the latest IBM 360 SRL Newsletter, GN20-0360, for the editions that are current and applicable.

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

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publications, Department 636, Neighborhood Road, Kingston, New York, 12401. All comments become the property of IBM.

© Copyright International Business Machines Corporation 1967, 1968, 1970, 1971, 1972

This book is divided into two parts. Section I, "Supervisor Services", provides explanations and aids for using the facilities available through the supervisor by means of the macro instructions described in Section II, "Macro Instructions".

Section I is divided into five topics: "Program Management," "Task Creation," "Task Management," "Program Management Services," and "Main-Storage Management."

The "Program Management" section describes linkage conventions and ways that the supervisor can assist you in linking together the separate pieces of your program.

The "Task Creation" section describes how the system creates a task for you, and how you can create other tasks (under MVT or under MFT with subtasking). Basically, it tells how to use the ATTACH macro instruction.

The "Task Management" section deals with communication among separate tasks and with synchronization of one task with another.

The "Program Management Services" section describes several miscellaneous services that you can use in your programs. It covers the ENQ and DEQ macro instructions, timer services, communication with the operator, abnormal termination and dumps, and other miscellaneous services.

The "Main-Storage Management" section describes how to acquire and release main storage, how to share it with other tasks, and how to specify which way it is to be divided into hierarchies.

Section II contains the descriptions and definitions of the supervisor macro instructions available in the IBM System/360 Operating System assembler language. It provides applications programmers coding the assembler language with the information necessary to code the macro instructions. The standard, list, and execute forms of the macro instructions are grouped, where applicable, for ease of reference.

Appendix A describes message routing procedures for multiple operator consoles.

Use of this book requires a basic knowledge of the operating system and of System/360 assembler language. Two books that contain information about these subjects are:

IBM System/360 Operating System

Introduction, GC28-6534

Assembler Language, GC28-6514

If you are using the MVT version of the control program with the time sharing option (TSO), note that this book also assumes that you understand how to use TSO. Specifically, the book assumes that you are familiar with the concepts discussed in the following books:

IBM System/360 Operating System

Time Sharing Option Command Language, GC28-6732, which describes the TSO command language that a terminal user must use to request computing services.

Time Sharing Option Guide, GC28-6698, which describes the concepts, features, and capabilities of TSO.

Time Sharing Option Guide to Writing a Terminal Monitor Program or a Command Processor, GC28-6764, which describes the programming features provided for user-written terminal monitor programs, command processors, and application programs.

If you are using the operating system without TSO, ignore the sections "Intercepting Abnormal Termination of Subtasks" and "Time Sharing Option (TSO) Services." Also ignore the TSO, PSB, and TJID operands of EXTRACT.

In the examples in Section I, the macro instructions are coded in just enough detail to make the examples clear. See Section II for a complete description of all the operands and options available with any of the macro instructions discussed here.

When other IBM manuals are referred to in the text, only partial titles are given. Here is a list of the complete titles and

order numbers of all manuals referred to in the text.

IBM System/360

Model 91 Functional Characteristics, GA22-6907

Model 195 Functional Characteristics, GA22-6943

Principles of Operation, GA22-6821

IBM System/360 Operating System

Advanced Checkpoint/Restart, GC28-6708

Job Control Language Reference, GC28-6704

Linkage Editor and Loader, GC28-6538

Programmer's Guide to Debugging, GC28-6670

Service Aids, GC28-6719

Storage Estimates, GC28-6551

Data Management Macro Instructions, GC28-6647

System Programmer's Guide, GC28-6550

IBM System/370

Principles of Operation, GA22-7000

SUMMARY OF CHANGES

Following is a list of programming changes that affect the information in this publication.

Release 20

Reason for Change	Items Changed or Added
Main Storage Hierarchy Support	ATTACH, GETMAIN, LINK, LOAD, XCTL
Time of Day Clock	TIME
II 272	EXTRACT
STAE Improvement	STAE
DXR Macro Instruction	DXR, extended-precision floating-point simulation

Release 20.1

Reason for Change	Items Changed or Added
Time Sharing Option (TSO)	ATTACH, DETACH, EXTRACT
ENQ Macro Instruction	Addition of RET=CHNG operand

Release 21

Reason for Change	Items Changed or Added
Generalized Trace Facility (GTF)	GTRACE, use of the facility
Multiple-line WTO Macro Instruction	WTO

CONTENTS

SECTION I: SERVICES 1
Introduction 1
Types of Services Available 1
Configurations of the Operating System 2

PROGRAM MANAGEMENT 3
Initial Requirements 3
Providing an Initial Base Register 3
Saving Registers 4
The SAVE Macro Instruction 5
Providing a Save Area 5
Establishing a Permanent Base Register 7
Linkage Registers 7
Acquiring the Information in the PARM Field of the EXEC Statement . . 7
Load Module Structure Types 8
Simple Structure 9
Planned Overlay Structure 9
Dynamic Structure 9
Load Module Execution 9
Passing Control in a Simple Structure 10
Passing Control Without Return 10
Initial Requirements 10
Passing Control 11
Passing Control with Return 12
Initial Requirements 12
Passing Control 12
Analyzing the Return 14
How Control is Returned 15
Return to the Control Program 17
Passing Control in a Planned Overlay Structure 17
Passing Control in a Dynamic Structure 17
Bringing the Load Module Into Main Storage 17
Load Module Location 17
The Search for the Load Module 19
Using an Existing Copy 22
Using the LOAD Macro Instruction 23
Passing Control With Return 24
The LINK Macro Instruction 24
Using the ATTACH Macro Instruction (MFT Without Subtasking) 26
Using CALL or Branch and Link 26
How Control is Returned 27
Passing Control Without Return 28
Passing Control Using a Branch Instruction 29
Using the XCTL Macro Instruction 29

TASK CREATION 31
Creating the Task 31
Task Priority 32
Priority of the Job Step Task 32
Priority of Subtasks 33
Time Slicing 34
MFT Systems Without Subtasking 34
MFT Systems With Subtasking 35
MVT Systems 36

TASK MANAGEMENT 37
Task and Subtask Communications 38
Task Synchronization 39
Manipulating Task Processing 39

PROGRAM MANAGEMENT SERVICES 40
Additional Entry Points 40
Entry Point and Calling Sequence Identifiers 41
Using a Serially Reusable Resource 41
Naming the Resource 42
Exclusive and Shared Requests 42
Processing the Request 43
Proper Use of ENQ and DEQ 44
Duplicate Requests 44
Releasing Control of the Resource 44
Conditional and Unconditional Requests 45
Avoiding Interlock 46
Obtaining Information From the Task Control Block 47
Timing Services 48
Date and Time of Day 49
Timing Services on the IBM System/370 49
Date and Time of Day 50
Interval Timing 50
Writing to One or More Operator Consoles 52
Writing to the Programmer 54
Writing to the Hard Copy Log 55
Writing to the System Log 55
Message Deletion 56
Operator Communication With A Problem Program 56
Generalized Trace Facility (GTF) Interface 57
Program Interruption Processing 58
Program Interruption Control Area 58
Program Interruption Element 59
Register Contents 60
Specifying an Attention Exit Routine 61
Precise and Imprecise Interruptions 61
Interruptions in the Models 91 and 195 61
Decimal Simulation in the Model 91 62
Extended-Precision Floating-Point Simulation 63
Extended Precision Division 64
Division Process 64
Arithmetic Exceptions 64
Calling the Simulator 66
Designing the EXIT Routine 66
Abnormal Condition Handling 69
Intercepting Abnormal Termination of Tasks 72
Intercepting Abnormal Termination of Subtasks 76
The DUMP 77
ABEND and SNAP Dumps 77
Indicative Dump 78
Core Image Dump 78

MAIN-STORAGE MANAGEMENT 79
Explicit Requests 79
Specifying Lengths 80
Types of Explicit Requests 80
Subpool Handling (in Mft Systems Without Subtasking) 81
Subpool Handling (in MFT Systems with Subtasking) 82
Subpool Handling (in MVT Systems) 82
Main Storage Control 82
Subpools in Task Communication 85
Implicit Requests 85
Load Module Management 85
Reenterable Load Modules 86
Reenterable Macro Instructions 86
Nonreenterable Load Modules 87
Releasing Main Storage 89
Storage Hierarchies 90

CHECKPOINT AND RESTART 91

SECTION II: MACRO INSTRUCTIONS 92
Introduction 92
Operating System Configurations and Options 92
Coding Aids 93
Writing the Macro Instructions 93
Continuation Lines 94
Additional Macro Instructions 95

STANDARD, LIST, AND EXECUTE FORMS 96
ABEND -- Abnormally Terminate a Job Step (MFT Without Subtasking) 97
ABEND -- Abnormally Terminate a Task (MVT, MFT With Subtasking) . 98
ATTACH -- Pass Control to a Program in Another Load Module (MFT Without Subtasking) 99
ATTACH -- Create a New Task (MFT With Subtasking) 101
ATTACH -- Create a New Task (MVT) 104
ATTACH -- List Form 109
ATTACH -- Execute Form 110
CALL -- Pass Control to a Control Section 112
CALL -- List Form 113
CALL -- Execute Form 114
CHAP -- Change Dispatching Priority (MFT Without Subtasking) . .115
CHAP -- Change Dispatching Priority (MVT, MFT With Subtasking) .116
CHKPT -- Take Checkpoint for Restart Within a Job Step 117
CHKPT -- List Form 120
CHKPT -- Execute Form 121
DELETE -- Relinquish Control of a Load Module 122
DEQ -- Release a Serially Reusable Resource 123
DEQ -- List Form 125
DEQ -- Execute Form 126
DETACH -- Delete a Subtask (MFT Without Subtasking) 127
DETACH -- Delete a Subtask (MFT With Subtasking) 128
DETACH -- Delete a Subtask (MVT) 129
DOM -- Delete Operator Message (Without the Multiple Console Support (MCS) Option) 130
DOM -- Delete Operator Message (With the Multiple Console Support (MCS) Option) 131
DXR -- Divide Extended Register 132
ENQ -- Request Control of a Serially Reusable Resource 133
ENQ -- List Form 136
ENQ -- Execute Form 137
EXTRACT -- Provide Information From TCB Fields (MFT Without Subtasking) 138
EXTRACT -- Provide Information From TCB Fields (MVT, MFT With Subtasking) 139
EXTRACT -- List Form 142
EXTRACT -- Execute Form 143
FREEMAIN -- Release Allocated Main Storage (MFT) 144
FREEMAIN -- Release Allocated Main Storage (MVT) 146
FREEMAIN -- List Form 148
FREEMAIN -- Execute Form 149
GETMAIN -- Allocate Main Storage (MFT) 150
GETMAIN -- Allocate Main Storage (MVT) 152
GETMAIN -- List Form 155
GETMAIN -- Execute Form 156
GTRACE -- Record Trace Data .157
GTRACE -- List Form 159
GTRACE -- Execute Form 160
IDENTIFY -- Add an Entry Point (MFT Without Identify Option) . .161
IDENTIFY -- Add an Entry Point (MFT With Identify Option, MVT). .162
LINK -- Pass Control to a Program in Another Load Module 164
LINK -- List Form 166

LINK -- Execute Form 167
LOAD -- Bring a Load Module Into Main Storage 169
POST -- Signal Event Completion 171
RETURN -- Return Control 172
SAVE -- Save Register Contents 173
SEGLD -- Load Overlay Segment and Continue Processing (MFT) . . .174
SEGLD -- Load Overlay Segment and Continue Processing (MVT) . . .175
SEGWT -- Load Overlay Segment and Wait 176
SNAP -- Dump Main Storage and Continue (MFT) 177
SNAP -- Dump Main Storage and Continue (MVT) 179
SNAP -- List Form 181
SNAP -- Execute Form 182
SPIE -- Specify Program Interruption Exit 184
SPIE -- List Form 186
SPIE -- Execute Form 187
STAE -- Specify Task Abnormal Exit 188
STAE -- List Form 191
STAE -- Execute Form 192
STATUS -- Change Subtask Status (MVT only) 193
STIMER -- Set Interval Timer (MFT Without Interval Timer Option) 194
STIMER -- Set Interval Timer (MFT With Interval Timer Option) . .195
STIMER -- Set Interval Timer (MVT) 197
TIME -- Provide Date (MFT Without Timer Option) 199
TIME -- Provide Time and Date (MFT With Timer Option, MVT) . . .200
TTIMER -- Test Interval Timer (MFT Without Interval Timer Option) 202
TTIMER -- Test Interval Timer (MFT With Interval Timer Option, MVT) 203
WAIT -- Wait for One or More Events 204
WAITR -- Wait for One or More Events 205
WTL -- Write to Log 206
WTL -- List Form 207
WTL -- Execute Form 208
WTO -- Write to Operator (Without Multiple Console Support) . . .209
WTO -- Write to Operator (With Multiple Console Support) 211
WTO -- List Form 213
WTO -- Execute Form 214
WTOR -- Write to Operator With Reply (Without Multiple Console Support) 215
WTOR -- Write to Operator With Reply (With Multiple Console Support) 216
WTOR -- List Form 217
WTOR -- Execute Form 218
XCTL -- Pass Control to a Program in Another Load Module 219
XCTL -- List Form 221
XCTL -- Execute Form 222

APPENDIX A: MESSAGE ROUTING FOR MULTIPLE OPERATOR CONSOLES 224
Routing Codes 224
Descriptor Codes 225
Operands for Use by the System Programmer 226
Summary of Operands 230

INDEX 235

Figure 1. Summary of characteristics and available options 2
Figure 2. Control section addressability 3
Figure 3. Internal entry point addressability 4
Figure 4. Save area format 4
Figure 5. Saving a range of registers 5
Figure 6. Saving registers 5-10, 14, and 15 5
Figure 7. Nonreenterable save area chaining 6
Figure 8. Reenterable save area chaining 6
Figure 9. Acquiring PARM field information 8
Figure 10. Load module characteristics 9
Figure 11. Passing control in a simple structure 11
Figure 12. Passing control with a parameter list 12
Figure 13. Passing control with return 13
Figure 14. Passing control with CALL 13
Figure 15. Test for normal return 15
Figure 16. Return code test using branching table 15
Figure 17. Establishing a return code 16
Figure 18. Use of the RETURN macro instruction 16
Figure 19. RETURN macro instruction with flag 17
Figure 20. Search for module, EP or EPLOC operands with DCB=0 or DCB operand omitted 19
Figure 21. Search for module, EP or EPLOC operands with DCB operand specifying private library 20
Figure 22. Search for module using DE operand 22
Figure 23. Use of the LINK macro instruction with the job or link library 25
Figure 24. Use of the LINK macro instruction with a private library 26
Figure 25. Use of the BLDL macro instruction 26
Figure 26. The LINK macro instruction with a DE operand 26
Figure 27. Misusing control program facilities 30
Figure 28. Determining partition dispatching priorities 35
Figure 29. Task hierarchy 37
Figure 30. Event control block 39
Figure 31. ENQ macro instruction processing 43
Figure 32. Interlock condition 47
Figure 33. Two requests for two resources 47
Figure 34. One request for two resources 47
Figure 35. Day of year processing 49
Figure 36. Interval timing 51
Figure 37. Writing to the operator 53
Figure 38. Writing to the operator with a reply 53
Figure 39. Using WTO and WTOR to write messages to the programmer . 54
Figure 40. Command input buffer contents 56
Figure 41. Specifying the GTRACE macro instruction 58
Figure 42. Program interruption control area 59
Figure 43. Use of the SPIE Macro Instruction 59
Figure 44. Program interruption element 60
Figure 45. Interruption code in the old program status word 62
Figure 46. Precise interruptions in IBM System/360 Models 65, 67, 75, 85, and 91, and System/370 Models 165 and 195 63
Figure 47. Summary of program interruptions 65
Figure 48. Calling the extended-precision floating-point simulator . 67
Figure 49. Return codes from the extended-precision floating-point simulator 68
Figure 50. Interruption codes returned by the simulator 69
Figure 51. Abnormal condition detection 70
Figure 52. Use of STAE macro instruction 73
Figure 53. Work area for STAE exit routine 75
Figure 54. Use of the GETMAIN macro instruction 81

Figure 55. Main-storage control 83
Figure 56. Using the list and the execute forms of the DEQ macro instruction 88
Figure 57. Summary of characteristics and available options 92
Figure 58. Continuation coding 94
Figure 59. DEQ macro instruction return codes 124
Figure 60. Location of return codes in main storage 124
Figure 61. ENQ macro instruction return codes 135
Figure 62. Location of return codes in main storage 135
Figure 63. EXTRACT answer area field order 141
Figure 64. Program Interruption Control Area 185
Figure 65. Routing/descriptor code combinations and resulting actions 209
Figure 66. Maximum 'text' field characters in a multiple-Line WTO message 212
Figure 67. Bit definitions for MSGTYP=Y 227
Figure 68. MCSFLAG parameters 228
Figure 69. ROUTCDE, DESC, and MSCTYP combinations 229
Figure 70. Summary of operands 230

INTRODUCTION

1.1 The supervisor provides the resources that your programs need in such a way that at any given time, as many resources as possible are in use. By using certain macro instructions, by specifying certain JCL parameters, and by organizing your program in certain ways, you can direct the supervisor as it goes about this job. This book tells you how to do it.

TYPES OF SERVICES AVAILABLE

1.2 The kinds of services you can request from the supervisor can be classified as follows:

- Program Management: Most programs are divided into segments. When these segments are separate load modules, the supervisor can be used to help them communicate with each other.

  The section of this book called "Program Management" discusses save areas, addressability, and passage of control from one segment of a program to another.

- Task Management: In some configurations of the operating system, units of work called tasks can compete with each other for resources.

  You can change your program's priority, break it into smaller units that compete with each other, and obtain certain information about how your tasks are progressing. The "Task Management" and "Task Creation" sections of this book tell you how.

- Main-Storage Management: Frequently, a program needs additional main storage for a particular requirement, such as for save areas. You can use main-storage management services to dynamically obtain additional main storage within your region or partition (an explicit request), and have the storage returned for other use when you no longer need it.

  When you request the use of a subtask or different load module, via the appropriate macro instruction, main storage management allocates space for the requested program (an implicit request) if sufficient main storage is available.

  The services available for obtaining and freeing, and for sharing main storage among several tasks are described in the "Main Storage Management" section of this book.

- Miscellaneous Services: The supervisor has facilities for providing dumps of main storage, communicating with the operator, handling abnormal conditions (such as program checks), allocating serially reusable resources, and timing events. The supervisor also provides a service for use with the time sharing option (TSO) which allows you to specify an attention exit routine. These services are discussed in the "Program Management Services" section.

CONFIGURATIONS OF THE OPERATING SYSTEM

1.3 This book covers two major configurations of the operating system: the operating system that provides multiprogramming with a fixed number of tasks (MFT), and the operating system that provides multiprogramming with a variable number of tasks (MVT). Unless otherwise indicated in the text, the descriptions in this section apply to all configurations of the operating system; when differences arise because of operating system options, these differences are explained.

1.4 A brief description of the configurations of the operating system is given in Figure 1. This table does not attempt to cover all of the options available in the operating system; it only summarizes the options that affect the material covered in this manual.

	MFT	MVT
Brief Description	Priority Scheduler, one (or, optionally, more than one) task per job step, 1 to 15 jobs processed concurrently	Priority Scheduler, one or more tasks per job step, 1 to 15 jobs processed concurrently
Attach	Optional	Standard
Identify	Optional	Standard
Timer	Optional	Standard
Interval Timer	Optional	Standard
Multiple Console Support	Optional	Optional
Time Sharing	Not Available	Optional

Figure 1. Summary of characteristics and available options

2.1 This section discusses the requirements for designing programs to be processed by the IBM System/360 Operating System. Included here are the procedures required when receiving control from the control program, the program design facilities available, and the conventions established for use in program management.

2.2 This discussion presents the conventions and procedures used by called and calling programs. Each program given control during the job step is initially a called program. During the execution of that program, the services of another program may be required, at which time the first program becomes a calling program. For example, the control program passes control to program A which is, at that point, a called program. During the execution of program A, control is passed to program B. Program A is now a calling program, program B a called program. Program B eventually returns control to program A, which eventually returns control to the control program. This is one of the simpler cases, of course. Program B could pass control to program C, which passes control to program D, which returns control to program C, etc. Each of these programs has the characteristics of either a called or calling program, regardless of whether it is the first, fifth or twentieth program given control during a job step.

2.3 The conventions and requirements that follow are presented in terms of one called and one calling program; these conventions and requirements apply to all called and calling programs in the system.

INITIAL REQUIREMENTS

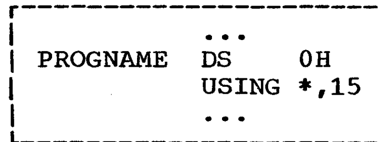
2.4 The following paragraphs discuss the procedures and conventions to be used when a program receives control from another program. Although the discussion is presented in terms of receiving control from the control program, the procedures and conventions apply as well when control is passed directly from another processing program. If the requirements presented here are followed in each of the programs used in a job step, the called program is not affected by the method used to pass control or by the identity of the program passing control.

PROVIDING AN INITIAL BASE REGISTER

2.5 When control is passed to your program from the control program, the address of the entry point in your program is contained in register 15. This address can be used to establish an initial base register, as shown in Figure 2 and Figure 3. In Figure 2, the entry point address is assumed to be the address of the first byte of the control section; an internal entry point is assumed in Figure 3. Since register 15 already contains the entry point address in both examples, no register loading is required.

```
PROGNAME  CSECT
          USING *,15
          ...
```

Figure 2. Control section addressability



```
         ...
PROGNAME DS    0H
         USING *,15
         ...
```

Figure 3. Internal entry point addressability

SAVING REGISTERS

2.6 The first action your program should take is to save the contents of the general registers. The contents of any register your program will modify must be saved, along with the contents of registers 0, 1, 14, and 15. The latter registers may be modified, along with the condition code, when system macro instructions are used to request data management or supervisor services.

2.7 The general registers are saved in an 18-word area provided by the control program; the format of this area is shown in Figure 4. When

Word	Contents
1	Used by PL/I language program
2	Address of previous save area (stored by calling program)
3	Address of next save area (stored by current program)
4	Register 14 (Return address)
5	Register 15 (Entry Point address)
6	Register 0
7	Register 1
8	Register 2
9	Register 3
10	Register 4
11	Register 5
12	Register 6
13	Register 7
14	Register 8
15	Register 9
16	Register 10
17	Register 11
18	Register 12

Figure 4. Save area format

control is passed to your program from the control program, the address of the save area is contained in register 13. As indicated in Figure 2, the contents of each of the registers must be saved at a predetermined location within the save area; for example, register 0 is always stored at word 6 of the save area, register 9 at word 15. The safest procedure is to save all of the registers; this ensures that later changes to your program will not result in the modification of the contents of a register that has not been saved.

2.8 To save the contents of the general registers, a store-multiple instruction, such as STM 14,12,12(13), can be written. This instruction places the contents of all the registers except register 13 in the proper words of the save area. (Saving the contents of register 13 is covered later.) If the contents of only registers 14, 15, and 0-6 are to be saved, the instruction would be STM 14,6,12(13).

THE SAVE MACRO INSTRUCTION

2.9 The SAVE macro instruction, provided to save you coding time, results in the instructions necessary to store a designated range of registers. An example of the use of the SAVE macro instruction is shown in Figure 5. The registers to be saved are coded in the same order as they would have been designated had an STM instruction been coded. A further use of the SAVE macro instruction is shown in Figure 6. The operand T specifies that the contents of registers 14 and 15 are to be saved in words 4 and 5 of the save area. The expansion of this SAVE macro instruction results in the instructions necessary to store registers 5-10, 14, and 15.

2.10 When you use the optional identifier name operand, you can code the SAVE macro instruction only at the entry point of a program. This is because the code resulting from the macro instruction with this operand requires that register 15 contain the address of the SAVE macro instruction.

PROVIDING A SAVE AREA

2.11 If any control section in your program is going to pass control to another control section and receive control back, your program is going to be a calling program and must provide another save area. Providing a save area allows the program you call to save registers without regard to whether it was called by your program, another processing program, or by the control program. If you establish beforehand what registers are available to the called program or control section, a save area is not necessary, but this is poor practice unless you are writing very simple routines.

```
PROGNAME  SAVE   (14,12)
          USING  PROGNAME,15
          ...
```

Figure 5. Saving a range of registers

```
PROGNAME  SAVE   (5,10),T
          USING  PROGNAME,15
          ...
```

Figure 6. Saving registers 5-10, 14, and 15

2.12 Whether or not your program is going to provide a save area, the address of the save area you used must be saved. You will need this address to restore the registers before you return to the program that called your program. If you are not providing a save area, you can keep the save area address in register 13, or save it in a fullword in your program. If you are providing another save area, the following procedure should be followed:

- Store the address of the save area you used (that is, the address passed to you in register 13) in the second word of the new save area.
- Store the address of the new save area (that is, the address you will pass in register 13) in the third word of the save area you used.

The reason for saving both addresses is discussed more fully under the heading "The Dump." Briefly, save the address of the save area you used so you can find the save area when you need it to restore the registers; save the address of the new save area so a trace from save area to save area is possible.

2.13 Figure 7 and Figure 8 show two methods of obtaining a new save area and of saving the save area addresses. In Figure 7, the registers are stored in the save area provided by the calling program (the control program). The address of this save area is then saved at the second word of the new save area, an 18 fullword area established through a DC instruction. Register 12 (any register could have been used) is loaded with the address of the previous save area. The address of the new save area is loaded into register 13, then stored at the third word of the old save area.

2.14 In Figure 8, the registers are again stored in the save area provided by the calling program. The entry point address in register 15 is loaded into register 5, which is declared as a base register. The contents of register 1 are saved in another register, and a GETMAIN

```
PROGNAME  STM    14,12,12(13)
          USING  PROGNAME,15
          ST     13,SAVEAREA+4
          LR     12,13
          LA     13,SAVEAREA
          ST     13,8(12)
          ...
SAVEAREA  DC     18A(0)
```

Figure 7. Nonreenterable save area chaining

```
PROGNAME  SAVE     (14,12)
          LR       5,15
          USING    PROGNAME,5
          LR       3,1
          GETMAIN  R,LV=72
          ST       13,4(1)
          ST       1,8(13)
          LR       13,1
          ...
```

Figure 8. Reenterable save area chaining

macro instruction is issued. The GETMAIN macro instruction (discussed in greater detail under the heading "Main Storage Management") requests the control program to allocate 72 bytes of main storage from an area outside your program, and to return the address of the area in register 1. The addresses of the new and old save areas are saved in the established locations, and the address of the new save area is loaded into register 13.

ESTABLISHING A PERMANENT BASE REGISTER

2.15 If your program does not use system macro instructions and does not pass control to another program, the base register established using the entry point address in register 15 is adequate. Otherwise, after you have saved your registers, establish base registers using one or more of registers 2-12. Register 15 is used by both the control program and your program for other purposes.

LINKAGE REGISTERS

2.16 Registers 0, 1, 13, 14, and 15 are known as the linkage registers, and are used in an established manner by the control program. It is good practice to use these registers in the same way in your program. As noted earlier, registers 0, 1, 14, and 15 may be modified when system macro instructions are used; registers 2-13 remain unchanged.

2.17 REGISTERS 0 AND 1: Registers 0 and 1 are used to pass parameters to the control program or to a called program. The expansion of a system macro instruction results in instructions required to load a value into register 0 or 1 or both, or to load the address of a parameter list into register 1. The control program also uses register 1 to pass parameters to your program or to the program you call. This is why the contents of register 1 were loaded into register 3 in Figure 8. For more information on parameter lists see "Passing Control", below.

2.18 REGISTER 13: Register 13 contains the address of the save area you have provided. The control program may use this save area when processing requests you have made using system macro instructions. A program you call can also use this save area when it issues a SAVE macro instruction.

2.19 REGISTER 14: Register 14 contains the return address of the program that called you, or an address within the control program to which you are to return when you have completed processing. The expansion of most system macro instructions results in an instruction to load register 14 with the address of your next sequential instruction. A BR 14 instruction at the end of any program will return control to the calling program as long as the contents of register 14 have not been altered.

2.20 REGISTER 15: Register 15, as you have seen, contains an entry point address when control is passed to a program from the control program. The entry point address should also be contained in register 15 when you pass control to another program. In addition, the expansions of some system macro instructions result in the instructions to load into register 15 the address of a parameter list to be passed to the control program. Register 15 is also used to pass a return code to a calling program.

ACQUIRING THE INFORMATION IN THE PARM FIELD OF THE EXEC STATEMENT

2.21 The manner in which the control program passes the information in the PARM field of your EXEC statement is a good example of how the control program uses a parameter register to pass information. When control is

passed to your program from the control program, register 1 contains the address of a fullword on a fullword boundary in your area of main storage (refer to Figure 9). The high order bit (bit 0) of this word is set to 1. This is a convention used by the control program to indicate the last word in a variable-length parameter list; you must use the same convention when making requests to the control program. The low-order three bytes of the fullword contain the address of a two-byte length field on a halfword boundary. The length field contains a binary count of the number of bytes in the PARM field, which immediately follows the length field. If the PARM field was omitted in the EXEC statement, the count is set to zero. To prevent possible errors, the count should always be used as a length attribute in acquiring the information in the PARM field. If your program is not going to use this information immediately, you should load the address from register 1 into one of registers 2-12 or store the address in a fullword in your program.

LOAD MODULE STRUCTURE TYPES

2.22 Each load module used during a job step can be designed in one of three load module structures: simple, planned overlay, or dynamic. A simple structure does not pass control to any other load modules during its execution, and is brought into main storage all at one time. A planned overlay structure may, if necessary, pass control to other load modules during its execution, and it is not brought into main storage all at one time. Instead, segments of the load module reuse the same area of main storage. A dynamic structure is brought into main storage all at one time, and passes control to other load modules during its execution. Each of the load modules to which control is passed can be one of the three structure types.

2.23 Figure 10 summarizes the characteristics of these load module structures.

2.24 The following paragraphs cover the advantages and disadvantages of each type of structure, and discuss the use of each.

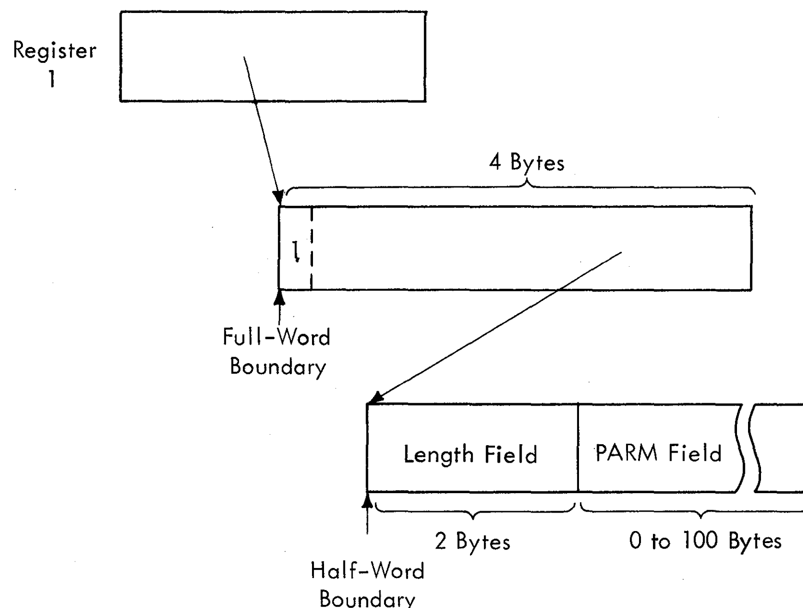


Figure 9. Acquiring PARM field information

Structure Type	Loaded All at One Time	Passes Control to Other Load Modules
Simple	Yes	No
Planned Overlay	No	Optional
Dynamic	Yes	Yes

Figure 10. Load module characteristics

SIMPLE STRUCTURE

2.25 A simple structure consists of a single load module produced by the linkage editor. The single load module contains all of the instructions required, and is brought into the main storage all at one time by the control program. The simple structure can be the most efficient of the three structure types because the instructions it uses to pass control do not require control program intervention. However, when a program is very large or complex, the main storage area required for the load module may exceed that which can be reasonably requested. (Main storage considerations are discussed under the heading "Main Storage Management.")

PLANNED OVERLAY STRUCTURE

2.26 A planned overlay structure consists of a single load module produced by the linkage editor. The entire load module is not brought into main storage at once; different segments of the load module use the same area of main storage. The planned overlay structure, while not as efficient as a simple structure in terms of execution speed, is more efficient than a dynamic structure. When using a planned overlay structure, control program assistance is required to locate and load portions of a single load module in a library; in a dynamic structure, many load modules in different libraries may need to be located and loaded in order to execute an equivalent program.

DYNAMIC STRUCTURE

2.27 A dynamic structure requires more than one load module during execution. Each load module required can operate as either a simple structure, a planned overlay structure, or another dynamic structure. The advantages of a dynamic structure over a planned overlay structure increase as the program becomes more complex, particularly when the logical path of the program depends on the data being processed. The load modules required in a dynamic structure are brought into main storage when required, and can be deleted from main storage when their use is completed.

LOAD MODULE EXECUTION

2.28 Depending on the configuration of the operating system and the macro instructions used to pass control, execution of the load modules is serial or in parallel. Execution of the load modules is always serial in an operating system with MFT without subtasking; there is only one task in the job step. Execution is also serial in an operating system

with MFT with subtasking or MVT, unless an ATTACH macro instruction is used to create a new task. The new task competes for control independently with all other tasks in the system. The load module named in the ATTACH macro instruction is executed in parallel with the load module containing the ATTACH macro instruction. The execution of the load modules is serial within each task.

2.29 The following paragraphs discuss passing control for serial execution of a load module. Creation and management of new tasks is discussed under the headings "Task Creation" and "Task Management."

PASSING CONTROL IN A SIMPLE STRUCTURE

2.30 There are certain procedures to follow when passing control to an entry point in the same load module. The established conventions to use when passing control are also discussed. These procedures and conventions provide the framework around which all program interface is built. Knowledge of the information contained in the section "Addressing -- Program Sectioning and Linking" in the *Assembler Language* publication is required.

PASSING CONTROL WITHOUT RETURN

2.31 A control section is usually written to perform a specific logical function within the load module. Therefore, there will be occasions when control is to be passed to another control section in the same load module, and no return of control is required. An example of this type of control section is a "housekeeping" routine at the beginning of a program which establishes values, initializes switches, and acquires buffers for the other control sections in the program. The following procedures should be used when passing control without return.

Initial Requirements

2.32 Because control will not be returned to this control section, you must restore the contents of register 14. Register 14 originally contained the address of the location in the calling program (for example, the control program) to which control is to be passed when your program is finished. Since the current control section will not make the return to the calling program, the return address must be passed to the control section that will make the return. In addition, the contents of registers 2-12 must be unchanged when your program eventually returns control, so these registers must also be restored.

2.33 If control were being passed to the next entry point from the control program, register 15 would contain the entry point address. You should use register 15 in the same way, so that the called routine remains independent of which program passed control to it.

2.34 Register 1 should be used to pass parameters. A parameter list should be established, and the address of the list placed in register 1. The parameter list should consist of consecutive full words starting on a fullword boundary, each fullword containing an *address* to be passed to the called control section in the three low order bytes of the word. The high-order bit of the last word should be set to 1 to indicate the last word of the list. The system convention is that the list contain addresses only. (The term "address parameters" is also used in this publication to describe entries in a parameter list.) You may, of course, deviate from this convention; however, when you deviate from any

system convention, you restrict the use of your programs to those programmers who are aware of your special conventions.

2.35 Since you have reloaded all the necessary registers, the save area that you used is now available, and can be reused by the called control section. You pass the address of the save area in register 13 just as it was passed to you. By passing the address of the old save area, you save the 72 bytes of main storage area required for a second, and unnecessary, save area.

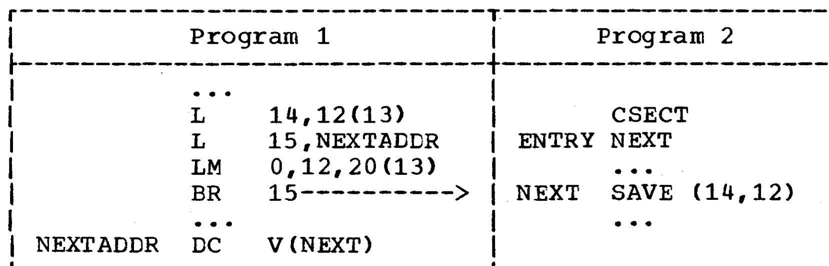
Passing Control

2.36 The common way to pass control between one control section and an entry point in the same load module is to load register 15 with a V-type address constant for the name of the external entry point, and then to branch to the address in register 15. The external entry point must have been identified using an ENTRY instruction in the called control section if the entry point is not the same as the control section name.

2.37 An example of proper register loading and control transfer is shown in Figure 11. In this example, no new save area is used, so register 13 still contains the address of the old save area. It is also assumed for this example that the control section will pass the same parameters it received to the next entry point. First, register 14 is reloaded with the return address. Next, register 15 is loaded with the address of the external entry point NEXT, using the V-type address constant at the location NEXTADDR. Registers 0-12 are reloaded, and control is passed by a branch instruction using register 15. The control section to which control is passed contains an ENTRY instruction identifying the entry point NEXT.

2.38 An example of the use of a parameter list is shown in Figure 12. Early in the routine the contents of register 1 (that is, the address of the fullword containing the PARM field address) were stored at the fullword PARMADDR. Register 13 is loaded with the address of the old save area, which had been saved in word 2 of the new save area. The contents of register 14 are restored, and register 15 is loaded with the entry point address.

2.39 The address of the list of parameters is loaded into register 1. These parameters include the addresses of two data control blocks (DCBs) and the original register 1 contents. The high-order bit in the last address parameter (PARMADDR) is set to 1 using an OR-immediate instruction. The contents of registers 2-12 are restored. (Since one of these registers was the base register, restoring the registers earlier would have made the parameter list unaddressable.) A branch instruction using register 15 passes control to entry point NEXT.



Program 1	Program 2
...	
L 14,12(13)	CSECT
L 15,NEXTADDR	ENTRY NEXT
LM 0,12,20(13)	...
BR 15---------->	NEXT SAVE (14,12)
...	...
NEXTADDR DC V(NEXT)	

Figure 11. Passing control in a simple structure

```
          ...
          USING  *,12              Establish addressability
EARLY     ST     1,PARMADDR        Save parameter address
          ...
          L      13,4(13)          Reload address of old save area
          L      14,12(13)         Load return address
          L      15,NEXTADDR       Load address of next entry point
          LA     1,PARMLIST        Load address of parameter list
          OI     PARMADDR,X'80'    Turn on last parameter indicator
          LM     2,12,28(13)       Reload remaining registers
          BR     15                Pass control
          ...
PARMLIST  DS     0A
DCBADDRS  DC     A(INDCB)
          DC     A(OUTDCB)
PARMADDR  DC     A(0)
NEXTADDR  DC     V(NEXT)
```

Figure 12. Passing control with a parameter list

PASSING CONTROL WITH RETURN

2.40 The control program passed control to your program, and your program will return control when it is through processing. Similarly, control sections within your program will pass control to other control sections, and expect to receive control back. An example of this type of control section is a "monitor" portion of a program; the monitor determines the order of execution of other control sections based on the type of input data. The following procedures should be used when passing control with return.

Initial Requirements

2.41 Registers 15 and 1 are used in exactly the same manner as they were used when control was passed without return. Register 15 contains the entry point address in the new control section and register 1 is used to pass a parameter list.

2.42 Using the standard convention, register 14 must contain the address of the location to which control is to be passed when the called control section completes processing. This time, of course, it is a location in the current control section. The address can be the instruction following the instruction which causes control to pass, or it can be another location within the current control section designed to handle all returns. Registers 2-12 are not involved in the passing of control; the called control section should not depend on the contents of these registers in any way.

2.43 You should provide a new save area for use by the called control section as previously described, and the address of that save area should be passed in register 13. Note that the same save area can be reused after control is returned by the called control section. One new save area is ordinarily all you will require regardless of the number of control sections called.

Passing Control

2.44 Two standard methods are available for passing control to another control section and providing for return of control. One is merely an extension of the method used to pass control without a return, and requires a V-type address constant and a branch or a branch and link

instruction. The other method uses the CALL macro instruction to provide a parameter list and establish the entry point and return point addresses. Using either method, the entry point must be identified by an ENTRY instruction in the called control section if the entry name is not the same as the control section name. Figure 13 and Figure 14 illustrate the two methods of passing control; in each example, it is assumed that register 13 already contains the address of a new save area.

2.45 Use of an inline parameter list and an answer area is also illustrated in Figure 13. The address of the external entry point is loaded into register 15 in the usual manner. A branch and link instruction is then used to branch around the parameter list and to load register 1 with the address of the parameter list. An inline parameter list such as the one shown in Figure 13 is convenient when you are debugging because the parameters involved are located in the listing (or the dump) at the point they are used, instead of at the end of the listing or dump. Note that the first byte of the last address parameter (ANSWERAD) is coded with the high-order bit set to 1 to indicate the end of the list. The area pointed to by the address in the ANSWERAD parameter is an area to be used by the called control section to pass parameters back to the calling control section. This is a possible method to use when a called control section must pass parameters back to the calling control section. Parameters are passed back in this manner so that no additional registers are involved. The area used in this example is twelve full words; the size of the area for any specific application depends on the requirements of the two control sections involved.

2.46 The CALL macro instruction in Figure 14 provides the same functions as the instructions in Figure 13. When the CALL macro instruction is expanded, the operands cause the following results:

NEXT
: A V-type address constant is created for NEXT, and the address is loaded into register 15.

```
          ...
          L     15,NEXTADDR    Entry point address in register 15
          CNOP  0,4
          BAL   1,GOOUT        Parameter list address in register 1
PARMLIST  DS    0A             Start of parameter list
DCBADDRS  DC    A(INDCB)       Input dcb address
          DC    A(OUTDCB)      Output dcb address
ANSWERAD  DC    B'10000000'    Last parameter bit on
          DC    AL3(AREA)      Answer area address
NEXTADDR  DC    V(NEXT)        Address of entry point
GOOUT     BALR  14,15          Pass control;  register 14 contains
                               return address
RETURNPT  ...   ...
AREA      DC    12F'0'         Answer area from NEXT
```

Figure 13. Passing control with return

```
          CALL  NEXT,(INDCB,OUTDCB,AREA),VL
RETURNPT  ...   ...
AREA      DC    12F'0'
```

Figure 14. Passing control with CALL

(INDCB,OUTDCB,AREA)
A-type address constants are created for the three parameters coded within parentheses, and the address of the first A-type address constant is placed in register 1.

VL
The high order bit of the last A-type address constant is set to 1.

2.47 Control is passed to NEXT using a branch and link instruction. The address of the instruction following the CALL macro instruction is loaded into register 14 before control is passed.

2.48 In addition to the results described above, the V-type address constant generated by the CALL macro instruction causes the load module with the entry point NEXT to be automatically edited into the same load module as the control section containing the CALL macro instruction. Refer to the Linkage Editor and Loader publication, if you are interested in finding out more about this service.

2.49 The parameter list constructed from the CALL macro instruction in Figure 14 contains only A-type address constants. A variation on this type of parameter list results from the following coding:

```
CALL NEXT,(INDCB,(6),(7)),VL
```

In the above CALL macro instruction, two of the parameters to be passed are coded as registers rather than symbolic addresses. The expansion of this macro instruction again results in a three-word parameter list; in this example, however, the expansion also contains the instructions necessary to store the contents of registers 6 and 7 in the second and third words, respectively, of the parameter list. The high-order bit in the third word is set to 1 after register 7 is stored. You can specify as many parameters as you need as address parameters to be passed, and you can use symbolic addresses or register contents as you see fit.

ANALYZING THE RETURN

2.50 When control is returned from the control program after processing a system macro instruction, the contents of registers 2-13 are unchanged. When control is returned to your control section from the called control section, registers 2-14 contain the same information they contained when control was passed, as long as system conventions are followed. The called control section has no obligation to restore registers 0 and 1; so the contents of these registers may or may not have been changed.

2.51 When control is returned, register 15 can contain a return code indicating the results of the processing done by the called control section. If used, the return code should be a multiple of 4, so a branching table can be used easily, and a return code of 0 should be used to indicate a normal return. The control program frequently uses this method to indicate the results of the requests you make using system macro instructions; an example of the type of return codes the control program provides is shown in the description of the IDENTIFY macro instruction in the macro instructions section.

2.52 The meaning of each of the codes to be returned must be agreed upon in advance. In some cases, either a "good" or "bad" indication (zero or nonzero) will be sufficient for you to decide your next action. If this is true, the code shown in Figure 15 could be used to analyze the results. Many times, however, the results and the alternatives are more complicated, and a branching table, such as shown in Figure 16, could be used to pass control to the proper routine.

```
RETURNPT  LTR  15,15    Test return code for zero
          BNZ  ERRORTN  Branch if not zero to error routine
          ...
```

Figure 15. Test for normal return

```
RETURNPT  B  RETTAB(15)  Branch to table using return code
RETTAB    B  NORMAL      Branch to normal routine
          B  COND1       Branch to routine for condition 1
          B  COND2       Branch to routine for condition 2
          B  GIVEUP      Branch to routine to handle
                           impossible situations
          ...
```

Figure 16. Return code test using branching table

HOW CONTROL IS RETURNED

2.53 In the discussion of the return under the heading "Analyzing the Return" it was indicated that the control section returning control must restore the contents of registers 2-14. Because these are the same registers reloaded when control is passed without a return, refer to the discussion under "Passing Control Without Return" for detailed information and examples. The contents of registers 0 and 1 do not have to be restored.

2.54 Register 15 can contain a return code when control is returned. As indicated previously, a return code should be a multiple of four with a return code of zero indicating a normal return. The return codes other than zero that you use can have any meaning, as long as the control section receiving the return codes is aware of that meaning.

2.55 The return address is the address originally passed in register 14; return of control should always be passed to that address. You can either use a branch instruction such as BR 14, or you can use the RETURN macro instruction. An example of each method of returning control is discussed in the following paragraphs.

2.56 Figure 17 is a portion of a control section used to analyze input data cards and to check for an out-of-tolerance condition. Each time an out-of-tolerance condition is found, in addition to some corrective action, one is added to the value at the address STATUSBY. After the last data card is analyzed, this control section returns to the calling control section, which proceeds based on the number of out-of-tolerance conditions encountered. The coding shown in Figure 17 causes register 13 to be loaded with the address of the save area this control section used, then reloads register 14 with the proper return address. The contents of register 15 are set to zero, and the value at the address STATUSBY (the number of errors) is placed in the low-order eight bits of the register. The contents of register 15 are shifted to the left two places to make the value a multiple of four. Registers 2-12 are reloaded, and control is returned to the address in register 14.

2.57 The RETURN macro instruction is provided to save coding time. The expansion of the RETURN macro instruction provides the instructions necessary to restore a designated range of registers, provide the proper return code value in register 15, and branch to the address in register 14. In addition, the RETURN macro instruction can be used to flag the save area used by the returning control section; this flag, a byte containing all ones, is placed in the high-order byte of word four of

the save area after the registers have been restored. The flag indicates that the control section that used the save area has returned to the calling control section. You will find that the flag is useful when tracing the flow of your program in a dump. For a complete record of program flow, a separate save area must be provided by each control section each time control is passed. This is usually not done because it requires too much main storage.

2.58 The contents of register 13 must be restored before the RETURN macro instruction is issued. The registers to be reloaded should be coded in the same order as they would have been designated had a load-multiple (LM) instruction been coded. You can load register 15 with the return code value before you code the RETURN macro instruction, you can specify the return code value in the RETURN macro instruction, or you can reload register 15 from the save area.

2.59 The code shown in Figure 18 provides the same result as the code shown in Figure 17. Registers 13 and 14 are reloaded, and the proper value is established in register 15. The RETURN macro instruction causes registers 2-12 to be reloaded, and control to be passed to the address in register 14. The save area used is not flagged. The RC=(15) operand indicates that register 15 already contains the return code value, and the contents of register 15 are not to be altered.

2.60 Figure 19 illustrates another use of the RETURN macro instruction. The correct save area address is again established, then the RETURN macro instruction is issued. In this example, registers 14 and 0-12 are reloaded, a return code of 8 is placed in register 15, the save area is flagged, and control is returned. Specifying a return code overrides the request to restore register 15 even though register 15 is within the designated range of registers.

```
          ...
          L    13,4(13)        Load address of previous save area
          L    14,12(13)       Load return address
          SR   15,15           Set register 15 to zero
          IC   15,STATUSBY     Load number of errors
          SLA  15,2            Set return code to multiple of 4
          LM   2,12,28(13)     Reload registers 2-12
          BR   14              Return
          ...
STATUSBY  DC   X'00'
```

Figure 17. Establishing a return code

```
          ...
          L         13,4(13)          Restore save area address
          L         14,12(13)         Return address in register 14
          SR        15,15             Zero register 15
          IC        15,STATUSBY       Load number of errors
          SLA       15,2              Set return code to multiple of 4
          RETURN    (2,12),RC=(15)    Reload registers and return
          ...
STATUSBY  DC        X'00'
```

Figure 18. Use of the RETURN macro instruction

```
...
L        13,4(13)
RETURN   (14,12),T,RC=8
```

Figure 19. RETURN macro instruction with flag

RETURN TO THE CONTROL PROGRAM

2.61 The discussion in the preceding paragraphs has covered passing control within one load module, and has been based on the assumption that the load module was brought into main storage because of the program name specified in the EXEC statement. The control program established only one task to be performed for the job step. When the logical end of the program is reached, control is returned to the address passed in register 14 to the first control section in the program. When the control program receives control at this point, it terminates the task it created for the job step, compares the return code in register 15 with any COND values specified on the JOB and EXEC statements, and determines whether or not the following job steps, if any, should be executed.

PASSING CONTROL IN A PLANNED OVERLAY STRUCTURE

2.62 A complete discussion of the requirements for passing control in an overlay environment is provided in the Linkage Editor and Loader manual.

PASSING CONTROL IN A DYNAMIC STRUCTURE

2.63 The discussion of passing control in a simple structure has provided the necessary background for the discussion of passing control in a dynamic structure. Within each load module, control should be passed as in a simple structure or planned overlay structure. If you can determine which control sections will make up a load module before you code the control sections and if they will fit in the main storage available, you should pass control within the load module without involving the control program. The macro instructions discussed in this section provide increased linkage capability, but they require control program intervention and possibly increased execution time.

BRINGING THE LOAD MODULE INTO MAIN STORAGE

2.64 The load module containing the entry point name you specified on the EXEC statement is automatically brought into main storage by the control program. Any other load modules you require during your job step are brought into main storage by the control program as a result of specific requests for dynamic acquisition; these requests are made through the use of the LOAD, LINK, ATTACH, or XCTL macro instructions. The following paragraphs discuss the proper use of these macro instructions.

LOAD MODULE LOCATION

2.65 Initially, each load module that you can obtain dynamically is located in a library (partitioned data set). This library is the link library, the job or step library, task library, or a private library.

2.66
- The link library is always present and is available to all job steps of all jobs. The control program provides the necessary data control block for the library, and logically connects the library to your program, making the members of the library available to your program.

2.67 • The job and step libraries are explicitly established by including //JOBLIB and //STEPLIB DD statements in the input stream. The //JOBLIB DD statement is placed immediately after the JOB statement, while the //STEPLIB DD statement is placed among the DD statements for a particular job step. The job library is available to all steps of your job, except those that have step libraries. A step library is available to a single job step; if there is a job library, the step library replaces the job library for the step. For either the job library or the step library, the control program provides the necessary data control block and issues the OPEN macro instruction to logically connect the library to your program.

2.68 • In systems with MVT, unique task libraries may be established by using the TASKLIB operand of the ATTACH macro instruction. The issuer of the ATTACH macro instruction is responsible for providing the DD statement and opening the data set or sets. If the TASKLIB operand is omitted, the task library of the attaching task is propagated to the attached task. In the following example, Task A's job library is LIB1. Task A attaches Task B, specifying TASKLIB=LIB2 on the ATTACH macro instruction. Task B's task library is therefore LIB2. When Task B attaches Task C, LIB2 is searched for Task C before LIB1 or the link library. Because Task B did not specify a unique task library for Task C, its own task library (LIB2) is propagated to Task C and will be the first library searched when Task C requests that a module be brought into main storage.

```
Task A        ATTACH EP=B,TASKLIB=LIB2
Task B        ATTACH EP=C
```

2.69 • A private library is established by including a DD statement in the input stream, and is available only to the job step in which it is defined. You must provide the necessary data control block and issue the OPEN macro instruction for each data set. You may use more than one private library by including more than one DD statement and associated data control block.

2.70 A library can be a single partitioned data set, or a collection of such data sets. When it is a collection, you define each data set by a separate DD statement, but you assign a name only to the statement that defines the first data set. Thus, a job library consisting of three partitioned data sets would be defined as follows:

```
//JOBLIB DD DSNAME=PDS1,---
//       DD DSNAME=PDS2,---
//       DD DSNAME=PDS3,---
```

The three data sets (PDS1, PDS2, PDS3) are processed as one, and are said to be <u>concatenated</u>. Concatenation and the use of partitioned data sets are discussed in more detail in the <u>Data Management Services</u> publication.

2.71 Operating systems with MFT or MVT may already have some of the load modules from the link library in main storage in an area called the resident reenterable module area (optional in MFT) or the link pack area (MVT). The contents of these areas are determined at Initial Program Loading time, and will vary depending on the requirements of your installation. In an operating system with MVT, the link pack area contains frequently used, reenterable load modules from the link library along with data management load modules; these load modules can be used by any job step in any job. When it is started, TSO extends the link pack area. In an operating system with MFT, the resident reenterable module area can contain user-written modules and the loader, discussed in the <u>Linkage Editor and Loader</u> publication, and all reenterable graphics subroutine package (GSP) modules.

2.72 With the exception of those load modules contained in this area, copies of all of the load modules you request are brought into your area of main storage, and are available to any task in your job step. For systems with MVT and MFT with subtasking, the portion of your area containing the copies of load modules is called the job pack area.

The Search for the Load Module

2.73 In response to your request for a copy of a load module, the control program searches the job pack area (MVT and MFT with subtasking), the libraries, and the link pack area (MVT) or the resident reenterable module area (MFT). If a copy of the load module is found in one of the pack areas, the control program determines whether or not that copy can be used, based on criteria discussed under the heading "Using an Existing Copy." If an existing copy can be used, the search stops. If it can not be used, the search continues until the module is located in a library. The load module is then brought into the job pack area.

2.74 The order in which the libraries and pack areas are searched depends on whether the system is MVT or MFT, and upon the operands used in the macro instruction requesting the load module. The operands that define the order of the search are the EP, EPLOC, DE, and DCB operands. The EP, EPLOC, and DE operands are used to specify the name of the entry point in the load module; you code one of the three every time you use a LINK, LOAD, XCTL, or ATTACH macro instruction. The DCB operand is used to indicate the address of the data control block for the library containing the load module, and is optional. Omitting the DCB operand or using the DCB operand with an address of zero specifies the data control blocks for the link library, the job or step library, or the task library.

2.75 The following paragraphs discuss the order of the search when the entry point name used is a member name.

2.76 The EP and EPLOC operands require the least effort on your part; you provide only the entry point name, and the control program searches for a load module having that entry point name. Figure 20 shows the order of the search when EP or EPLOC is coded, and the DCB operand is omitted or DCB=0 is coded.

MFT	MVT
The job pack area, the optional resident access method area, and the loaded program list are searched, in that order.	The job pack area of the region is searched for an available copy
	The requesting task's task library and all the unique task libraries of its direct ascendants are searched.
The resident reenterable load module area is searched (optional).	The step library is searched; if there is no step library, the job library (if any) is searched.
The step library or the job library (if any) is searched. If both libraries are specified, the job library is not searched.	The link pack area is searched.
The link library is searched.	The link library is searched.

Figure 20. Search for module, EP or EPLOC operands with DCB=0 or DCB operand omitted

2.77 When used without the DCB operand, the EP and EPLOC operands provide the easiest method of requesting a load module from the link, task, job, or step library. In a system with MVT, the task libraries are searched before the job or step library, beginning with the task library of the task that issued the request and continuing through the task libraries of all its ascendents. The job or step library is then searched, followed by the link library. In a system with MFT, the job or step library is the first searched, followed by the link library. The data sets that make up these libraries are searched in the order of their DD statements.

2.78 A job, step, or link library or a data set in one of these libraries can be used to hold one version of a load module, while another can be used to hold another version with the same entry point name. If one version is in the link library, you can ensure that the other will be found first by including it in the job or step library. However, if both versions are in the job or step library, you must define the data set that contains the version you want to use before that which contains the other version. For example, if the wanted version is in PDS1 and the unwanted version is in PDS2, a step library consisting of these data sets should be defined as follows:

```
//STEPLIB DD DSNAME=PDS1,---
//        DD DSNAME=PDS2,---
```

If, however, the first version in the job or step library has been previously loaded and the version in the link library or the second version in the job library is desired, the DCB operand must be coded on the macro instruction.

2.79 This is not the case for task libraries. Extreme caution should be used when specifying module names in unique task libraries, because duplicate names may lead to the wrong module being given to the task requesting that the module be brought into main storage. Once a module has been loaded, the module name is known to all tasks in the region and a copy of that module will be given to all tasks requesting that that module name be loaded, regardless of the requester's task library.

2.80 If you know that the load module you are requesting is a member of one of the private libraries, you can still use the EP or EPLOC operands, this time in conjunction with the DCB operand. You would specify the address of the data control block for the private library in the DCB operand. The order of the search for EP or EPLOC with the DCB operand is shown in Figure 21.

MFT	MVT
The partition is searched.	The job pack area of the region is searched for an available copy.
The resident reenterable load module area is searched (optional).	The specified library is searched.
The specified library is searched.	The link pack area is searched.
	The link library is searched.

Figure 21. Search for module, EP or EPLOC operands with DCB operand specifying private library

2.81 Searching a job step, or task library slows the retrieval of load modules from the link library; to speed this retrieval, you should limit the size of the job and step libraries. You can best do this by eliminating the job library altogether, and providing step libraries where required. You can limit each step library to the data sets required by a single step; some steps (such as compile) will not require a step library, and therefore will not require any unnecessary search in retrieving modules from the link library. For maximum efficiency, you should define a job library only when a step library would be required for every step, and every step library would be the same.

2.82 The DE operand requires more work than the EP and EPLOC operands, but it can reduce the amount of time spent searching for a load module. Before you can use this operand, you must use the BLDL macro instruction to obtain the directory entry for the module. The directory entry is part of the library that contains the module.

2.83 To save time, the BLDL macro instruction used must obtain directory entries for more than one entry point name. You specify the names of the load modules and the address of the data control block for the library when using the BLDL macro instruction; the control program places a copy of the directory entry for each entry point name requested in a designated location in main storage. If no DCB address is given, the task library, job/step library, and link library are searched. If you specify the link library and the job or step library, the directory information indicates from which library the directory entry was taken. The directory entry always indicates the exact relative track and block location of the load module in the library. If the load module is not located on the library you indicate, a return code is given. You can then issue another BLDL macro instruction specifying a different library. For information about the use of load modules by more than one task, see the description of BLDL in OS Data Management Services Guide.

2.84 To use the DE operand, you provide the address of the directory entry, and code or omit the DCB operand to indicate the same library specified in the BLDL macro instruction. The order of the search when the DE operand is used is shown in Figure 22 for the link, job, step, and private libraries.

2.85 The preceding discussion of the search is based on the premise that the entry point name you specified is the member name. When you are using an operating system with MFT, the same search results from specifying an alias rather than a member name. When you are using an operating system that includes MVT, the control program checks if the entry point name is an alias when the load module is found in a library. If the name is an alias, the control program obtains the corresponding member name from the library directory, then searches the link pack and job pack areas using the member name to determine if a usable copy of the load module exists in main storage. If a usable copy does not exist in a pack area, a new copy is brought into the job pack area. Otherwise, the existing copy is used, conserving main storage and eliminating the loading time.

2.86 As the discussion of the search indicates, you should choose the operands for the macro instruction that provide the shortest search time. The search of a library actually involves a search of the directory, followed by copying the directory entry into main storage, followed by loading the load module into main storage. If you know the location of the load module, you should use the operands in your macro instruction that eliminate as many of these unnecessary searches as possible, as indicated in Figure 20, Figure 21, and Figure 22. Examples of the use of these figures are shown in the discussion of passing control.

2.87

MFT	MVT
Directory Entry Indicates Link Library and DCB=0 or DCB Operand Omitted	
The partition is searched.	The job pack area for the region is searched for an available copy.
The resident reenterable load module area is searched (optional).	The link pack area is searched.
The module is obtained from the link library.	The module is obtained from the link library.
Directory Entry Indicates Job, Step, or Task Library and DCB=0 or DCB Operand Omitted	
The job pack area for the partition is searched for an available copy.	The job pack area for the region is searched for an available copy.
The module is obtained from the step library; if there is no step library, the module is obtained from the job library.	The module is obtained from the library indicated in the directory entry.
DCB Operand Indicates Private Library	
The job pack area for the partition is searched for an available copy.	The job pack area for the region is searched for an available copy.
The module is obtained from the specified private library.	The module is obtained from the specified private library.

Figure 22. Search for module using DE operand

Using an Existing Copy

2.87 The control program will use a copy of the load module already in the link pack area or job pack area if the copy can be used. Whether the copy can be used or not depends on the reusability and current status of the load module; that is, the load module attributes, as designated using linkage editor control statements, and whether or not the load module has already been used or is in use. The status information is available to the control program only when you specify the load module entry point name on an EXEC statement, or when you use ATTACH, LINK, or XCTL macro instructions to transfer control to the load module. The control program will protect you from obtaining an unusable copy of a load module as long as you always "formally" request a copy using these macro instructions (or the EXEC statement); if you ever pass control in any other manner (for instance, a branch or a CALL macro instruction), the control program, because it is not informed, cannot protect you. Note that attributes can be dynamically changed by using the IDENTIFY macro instruction (see "Additional Entry Points").

2.88 Operating System With MVT: If you are using an operating system with MVT, all reenterable modules (modules designated as reenterable using the linkage editor) from any library are completely reusable; only one copy is ever placed in the link pack area or brought into your job pack area, and you get immediate control of the load module. If the module is serially reusable, only one copy is ever placed in the job pack area; this copy will always be used for a LOAD macro instruction. If the copy

is in use, however, and the request is made using a LINK, ATTACH, or XCTL macro instruction, the task requiring the load module is placed in a wait condition until the copy is available. A LINK macro instruction should not be issued for a serially reusable load module currently in use for the same task; the task will be abnormally terminated. (This could occur if an exit routine issued a LINK macro instruction for a load module in use by the main program.)

2.89 If the load module is nonreusable, a LOAD macro instruction will always bring in a new copy of the load module; an existing copy is used only if a LINK, ATTACH, or XCTL macro instruction is issued and the copy has not been used previously. Remember, the control program can determine if a load module has been used or is in use only if all of your requests are made using LINK, ATTACH, or XCTL macro instructions.

2.90 MFT Systems With Subtasking: If you are using an MFT system with subtasking, the LOAD macro instruction enables all tasks in a partition to share the same copy of a reenterable module invoked by a previous LOAD macro instruction. (A reenterable module in MFT is considered reusable only.) If the reenterable module is again invoked by a LINK, XCTL, or ATTACH macro instruction and a previous request is still active, a new copy of the module will be brought into main storage.

2.91 MFT Systems Without Subtasking: If you are using an operating system with MFT, the macro instruction used to request the load module also determines if an existing copy can be used. If a LOAD macro instruction is issued, an existing copy is always used to satisfy the request, without regard to the reusability designation or the current status of the copy. However, if an ATTACH, LINK, or XCTL macro instruction is issued, an existing copy is used only if that copy was brought into main storage as a result of a request using a LOAD macro instruction and the copy is not in use; otherwise, a new copy is brought into the job pack area.

2.92 MFT Systems With the Resident Reenterable Module Area Option: If you are using an operating system with the MFT resident reenterable module area option, and you request use of a module by issuing an ATTACH, LINK, LOAD, or XCTL macro instruction, the supervisor will search the resident reenterable module area for a copy of the module before fetching a new copy into main storage.

Using the LOAD Macro Instruction

2.93 The LOAD macro instruction is used to ensure that a copy of the specified load module is in main storage in your job pack area if it is not preloaded into the link pack area. When a LOAD macro instruction is issued, the control program searches for the load module as discussed previously, and brings a copy of the load module into the job pack area if required. When the control program returns control, register 0 contains the main storage address of the entry point specified for the requested load module. Normally, the LOAD macro instruction is used only for a reenterable or serially reusable load module, since the load module is retained even though it is not in use.

2.94 The control program also establishes a "responsibility" count for the copy, and adds one to the count each time the requirements of a LOAD macro instruction are satisfied by the same copy. As long as the responsibility count is not zero, the copy is retained in main storage.

2.95 The responsibility count for the copy is lowered by one when a DELETE macro instruction is issued during the task which was active when the LOAD macro instruction was issued. When a task is terminated, the count is lowered by the number of LOAD macro instructions issued for the copy when the task was active minus the number of deletions.

2.96 When the responsibility count for a copy in a job pack area reaches zero, the main storage area containing the copy is made available; the copy is never reused after the responsibility count established by LOAD macro instructions reaches zero.

2.97 Copies of load modules are not added to or deleted from the link pack area; LOAD and DELETE macro instructions issued for load modules already in the link pack area result in returns indicating successful completion, however.

PASSING CONTROL WITH RETURN

2.98 The LINK macro instruction is used to pass control between load modules and to provide for return of control. In an operating system with MFT without subtasking, the ATTACH macro instruction is executed in a similar manner to the LINK macro instruction. You can also pass control using branch or branch and link instructions or the CALL macro instruction; however, when you pass control in this manner you must protect against multiple uses of nonreusable or serially reusable modules. The following paragraphs discuss the requirements for passing control with return in each case.

The LINK Macro Instruction

2.99 When you use the LINK macro instruction, as far as the logic of your program is concerned, you are passing control to another load module. Remember, however, that you are requesting the control program to assist you in passing control. You are actually passing control to the control program, using an SVC instruction, and requesting the control program to find a copy of the load module and pass control to the entry point you designate. There is some similarity between passing control using a LINK macro instruction and passing control using a CALL macro instruction in a simple structure. These similarities are discussed first.

2.100 The convention regarding registers 2-12 still applies; the control program does not change the contents of these registers, and the called load module should restore them before control is returned. You must provide the address in register 13 of a save area for use by the called load module; the control program does not use this save area. You can pass address parameters in a parameter list to the load module using register 1; the LINK macro instruction provides the same facility for constructing this list as the CALL macro instruction. Register 0 is used by the control program and the contents will be modified.

2.101 There is also some difference between passing control using a LINK macro instruction and passing control using a CALL macro instruction. When you pass control in a simple structure, register 15 contains the entry point address and register 14 contains the return point address. When the called load module gets control, that is still what registers 14 and 15 contain, but when you use the LINK macro instruction, it is the control program that establishes these addresses. When you code the LINK macro instruction, you provide the entry point name and possibly some library information using the EP, EPLOC, or DE, and DCB operands. But you have to get this entry point and library information to the control program. The expansion of the LINK macro instruction does this, by creating a control program parameter list (the information required by the control program) and placing the address of this parameter list in register 15. After the control program finds the entry point, it places the address in register 15.

2.102 The return address in your control section is always the instruction following the LINK; that is not, however, the address that the called load module receives in register 14. The control program saves the address of the location in your program in its own save area, and places in register 14 the address of a routine within the control program that will receive control. Because control was passed using the control program, return must also be made using the control program.

2.103 The control program establishes a responsibility count for a load module when control is passed using the LINK macro instruction. This is a separate responsibility count from the count established for LOAD macro instructions, but it is used in the same manner. The count is increased by one when a LINK macro instruction is issued, and decreased by one when return is made to the control program or when the called load module issues an XCTL macro instruction.

2.104 Figures 23 and 24 show the coding of a LINK macro instruction used to pass control to an entry point in a load module. In Figure 23, the load module is from the link, job, or step library; in Figure 24, the module is from a private library. Except for the method used to pass control, this example is similar to Figures 13 and 14. A problem program parameter list containing the addresses INDCB, OUTDCB, and AREA is passed to the called load module; the return point is the instruction following the LINK macro instruction. A V-type address constant is not generated, because the load module containing the entry point NEXT is not to be edited into the calling load module. Note that the EP operand is chosen, since the search begins with the job pack area and the appropriate library as shown in Figure 20.

2.105 Figures 25 and 26 show the use of the BIDL and LINK macro instructions to pass control. Assuming control is to be passed to an entry point in a load module from the link library, a BLDL macro instruction is issued to bring the directory entry for the member into main storage. (Remember, however, that time is saved only if more than one directory entry is requested in a BLDL macro instruction. Only one is requested here for simplicity. Time is also saved if the program links to the load module more than once, using the information in directory entry each time.)

2.106 The first operand of the BLDL macro instruction is a zero, which indicates that the directory entry is on the link or job library. The second operand is the address in main storage of the list description field for the directory entry. The first two bytes at LISTADDR indicate the number of directory entries in the list; the second two bytes indicate the length of each entry. If the entry is to be used in a LINK, LOAD, ATTACH, or XCTL macro instruction, the entry must be 58 bytes in length. A character constant is established to contain the directory information to be placed there by the control program as a result of the BLDL macro instruction. The LINK macro instruction in Figure 26 can now be written. Note that the DE operand refers to the name field, not the list description field, of the directory entry.

```
          LINK  EP=NEXT,PARAM=(INDCB,OUTDCB,AREA),VL=1
RETURNPT  ...
AREA      DC    12F'0'
```

Figure 23. Use of the LINK macro instruction with the job or link library

```
         OPEN   (PVTLIB)
         ...
         LINK   EP=NEXT,DCB=PVTLIB,PARAM=(INDCB,OUTDCB,AREA),VL=1
         ...
PVTLIB   DCB    DDNAME=PVTLIBDD,DSORG=PO,MACRF=(R)
```

Figure 24. Use of the LINK macro instruction with a private library

```
           BLDL   0,LISTADDR
           ...
           DS     0H             List description field:
 LISTADDR  DC     H'01'             Number of list entries
           DC     H'58'             Length of each entry
 NAMEADDR  DC     CL8'NEXT'      Member name
           DS     25H            Area required for directory information
```

Figure 25. Use of the BLDL macro instruction

```
LINK   DE=NAMEADDR,DCB=0,PARAM=(INDCB,OUTDCB,AREA),VL=1
```

Figure 26. The LINK macro instruction with a DE operand

Using the ATTACH Macro Instruction (MFT Without Subtasking)

2.107 In a system without subtasking, the ATTACH macro instruction performs exactly the same functions as the LINK macro instruction and should be used in the same way. In a system with subtasking, however, you use the ATTACH macro instruction to cause parallel execution.

2.108 You should use the ATTACH macro instruction only when coding for upward compatibility with a system that includes subtasking. There are two additional operands provided with the ATTACH macro instruction: the ECB and ETXR operands. When used in an operating system with MVT or with MFT with subtasking, these operands provide a means of communication between tasks from the same job step. Refer to "Task Management" for a discussion of the ECB and ETXR operands.

Using CALL or Branch and Link

2.109 You can save time by passing control to a load module without using the control program. Passing control without using the control program is performed as follows: issue a LOAD macro instruction to obtain a copy of the load module, preceded by a BLDL macro instruction if you can shorten the search time by using it. The control program returns the address of the entry point in register 0. Load this address into register 15. The linkage requirements are the same when passing control between load modules as when passing control between control sections in the same load module: register 13 must contain a save area address, register 14 must contain the return point address, and register 1 is used to pass parameters in a parameter list. A branch instruction, a branch and link instruction, or a CALL macro instruction can be used to pass control, using register 15. The return will be made directly to you.

2.110 Note: When control is passed to a load module without using the control program, you must check the load module attributes and current status of the copy yourself, and you must check the current status in all succeeding uses of that load module during the job step, even when the control program is used to pass control.

2.111 The reason you have to keep track of the usability of the load module has been discussed previously: you are not allowing the control program to determine whether you can use a particular copy of the load module. The following paragraphs discuss your responsibilities when using load modules with various attributes. You must always know what the reusability attribute of the load module is. If you do not know, you should not attempt to pass control yourself.

2.112 If the load module is reenterable, one copy of the load module is all that is ever required for a job step. You do not have to determine the current status of the copy; it can always be used. The best way to pass control is to use a CALL macro instruction or a branch or branch and link instruction.

2.113 If the load module is serially reusable, one use of the copy must be completed before the next use begins. If your job step consists of only one task, preventing simultaneous use of the same copy involves making sure that the logic of your program does not require a second use of the same load module before completion of the first use. An exit routine must not require the use of a serially reusable load module also required in the main program.

2.114 Preventing simultaneous use of the same copy when you have more than one task in the job step requires more effort on your part. You must still be sure that the logic of the program for each task does not require a second use of the same load module before completion of the first use. You must also be sure that no more than one task requires the use of the same copy of the load module at one time; the ENQ macro instruction can be used for this purpose. Properly used, the ENQ macro instruction prevents the use of a serially reusable resource, in this case a load module, by more than one task at a time. Refer to "Program Management Services" for a complete discussion of the ENQ macro instruction. A conditional ENQ macro instruction can also be used to check for simultaneous use of a serially reusable resource within one task.

2.115 If the load module is nonreusable, each copy can only be used once; you must be sure that you use a new copy each time you require the load module. If you are using an operating system with MVT or with MFT with subtasking, you can ensure that you always get a new copy by using a LINK macro instruction or by doing as follows:

- Issue a LOAD macro instruction before you pass control.
- Pass control using a branch or a branch and link instruction or a CALL macro instruction only.
- Issue a DELETE macro instruction as soon as you are through with the copy.

2.116 If you are using an operating system with MFT without subtasking, you should perform the same three steps indicated above, and also make sure that you do not require a second use of the load module before completion of the first use.

HOW CONTROL IS RETURNED

2.117 The return of control between load modules is exactly the same as return of control between two control sections in the same load module.

The program in the load module returning control is responsible for restoring registers 2-14, possibly establishing a return code in register 15, and passing control using the address in register 14. The program in the load module to which control is returned can expect the contents of registers 2-13 to be unchanged, the contents of register 14 to be the return point address, and optionally, the contents of register 15 to be a return code. The return of control can be made using a branch instruction or the RETURN macro instruction. If control was passed without using the control program, that is all there is to it. However, if control was originally passed using the control program, the return of control is to the control program, then to the calling program. The action taken by the control program is discussed in the following paragraphs.

2.118 When control was passed using a LINK or ATTACH macro instruction, the responsibility count was increased by one for the copy of the load module to which control was passed to ensure that the copy would be in main storage as long as it was required. The return of control indicates to the control program that this use of the copy is completed, so the responsibility count is decremented by one. If you are using an operating system with MFT, the main storage area containing the copy is made available when the responsibility count reaches zero. If you are using an operating system with MVT, the copy is retained when the responsibility count reaches zero if all three of the following requirements are met:

- The load module attributes are serially reusable or reenterable.
- The count was not reduced to zero because of a DELETE macro instruction.
- The main storage area is not required for other purposes.

2.119 If control was originally passed using an ATTACH macro instruction (MFT without subtasking), the control program takes the following action:

- If the ECB operand was specified, the control program posts the return code in the indicated fullword.
- If the ETXR operand was specified, the control program passes control to the designated address, using register 15 to contain the entry point address, and register 14 to contain the return point address (to the control program). When the exit routine returns control, the control program passes control to the instruction following the ATTACH macro instruction without modifying the contents of any register except register 14. Register 15 does not, in this case, contain the return code.

2.120 If the ETXR operand was not specified, or if the LINK macro instruction was used to pass control, the control program only places the return point address into register 14, and passes control to that address. No other register contents are modified.

PASSING CONTROL WITHOUT RETURN

2.121 The XCTL macro instruction is used to pass control between load modules when no return of control is required. You can also pass control using a branch instruction; however, when you pass control in this manner, you must protect against multiple uses of non-reusable or serially reusable modules. The following paragraphs discuss the requirements for passing control without return in each case.

PASSING CONTROL USING A BRANCH INSTRUCTION

2.122 The same requirements and procedures for protecting against reuse of a nonreusable copy of a load module apply when passing control without return as were stated under "Passing Control With Return." The procedures for passing control are as follows.

2.123 A LOAD macro instruction should be issued to obtain a copy of the load module. The entry point address returned in register 0 is loaded into register 15. The linkage requirements are the same when passing control between load modules as when passing control between control sections in the same load module; register 13 must be reloaded with the old save area address, then registers 14 and 2-12 restored from that old save area. Register 1 is used to pass parameters in a parameter list. A branch instruction is issued to pass control to the address in register 15.

2.124 Mixing branch instructions and XCTL macro instructions is hazardous. The next topic explains why.

USING THE XCTL MACRO INSTRUCTION

2.125 The XCTL macro instruction, in addition to being used to pass control, is also used to indicate to the control program that this use of the load module containing the XCTL macro instruction is completed. Because control is not to be returned, the address of the old save area must be reloaded into register 13. The return point address must be loaded into register 14 from the old save area, as must the contents of registers 2-12. The XCTL macro instruction can be written to request the loading of registers 2-12, or you can do it yourself. If you restore all registers yourself, do not use the EP parameter. This creates an inline parameter list that needs your base register to be addressable, and your base register is no longer valid. If EP is used, you must have XCTL restore the base register for you.

2.126 When using the XCTL macro instruction, you pass parameters in a parameter list, with the address of the list contained in register 1. In this case, however, the parameter list must be established in a portion of main storage outside the current load module containing the XCTL macro instruction. This is because the copy of the current load module may be deleted before the called load module can use the parameters, as explained in more detail below.

2.127 The XCTL macro instruction is similar to the LINK macro instruction in the method used to pass control: control is passed by way of the control program using a control program parameter list. The control program loads a copy of the load module, if necessary, establishes the entry point address in register 15, saves the address passed in register 14 and replaces it with a new return point address within the control program, and passes control to the address in register 15. The control program adds one to the responsibility count for the copy of the load module to which control is to be passed, and subtracts one from the responsibility count for the current load module. The current load module in this case is the load module last given control using the control program in the performance of the active task. If you have been passing control between load modules without using the control program, chances are the responsibility count will be lowered for the wrong load module copy. And remember, when the responsibility count of a copy reaches zero, that copy may be deleted, causing unpredictable results if you try to return control to it.

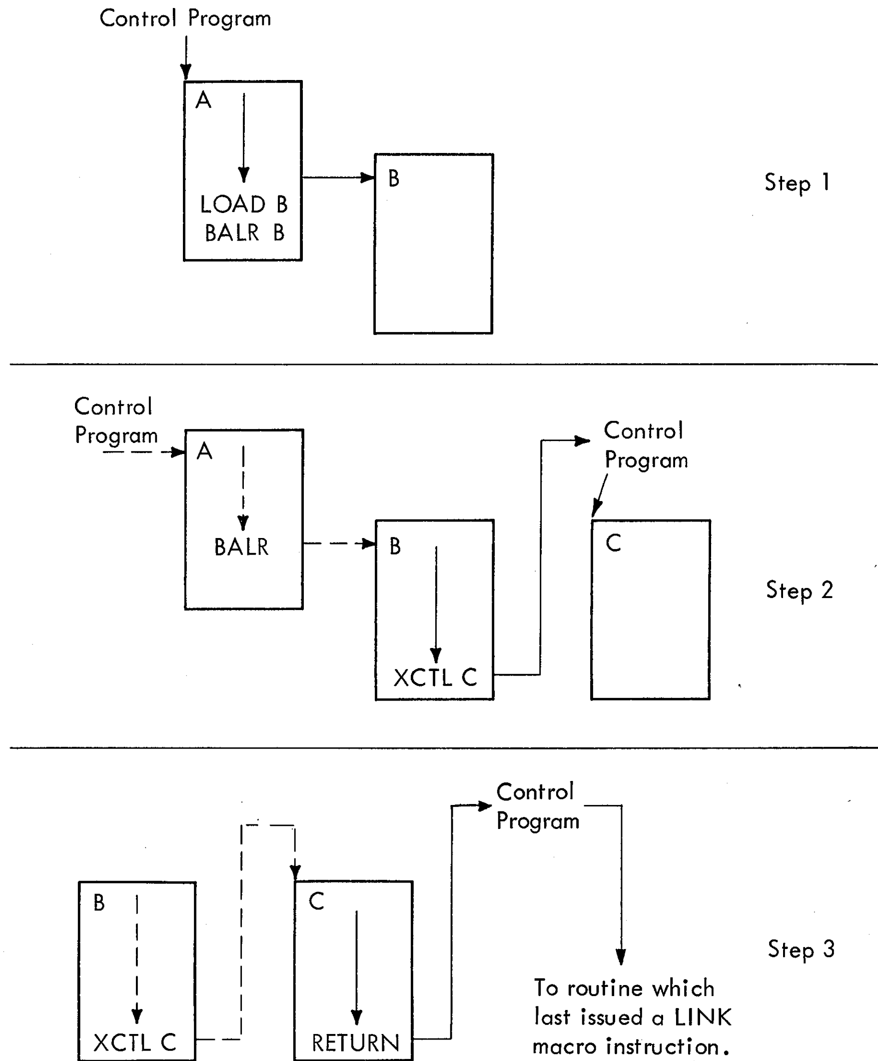


Figure 27. Misusing control program facilities

2.128 Figure 27 shows how this could happen. Control is given to load module A, which passes control to load module B (step 1) using a LOAD macro instruction and a branch and link instruction. Register 14 at this time contains the address of the instruction following the branch and link. Load module B then is executed, independent of how control was passed, and issues an XCTL macro instruction when it is finished (step 2) to pass control to load module C. The control program, knowing only of load module A, lowers the responsibility count of A by one, resulting in its deletion. Load module C is executed and returns to the address which used to follow the branch and link instruction. Step 3 of Figure 27 indicates the result.

2.129 Two methods are available for ensuring that the proper responsibility count is lowered. One way is to always use the control program to pass control with or without return. The other method is to use only LOAD and DELETE macro instructions to determine whether or not a copy of a load module should remain in main storage.

3.1 In any configuration of the operating system, one task is created by the control program as a result of initiating execution of the job step. In an operating system with MFT without subtasking, only the control program can create tasks; your program cannot create tasks.

3.2 In an operating system with MVT or with MFT with subtasking, you can create additional tasks in your program. If you do not, however, the job step task is the only task in a job being executed under MVT or under MFT with subtasking. The benefits of a multiprogramming environment are still available even with only one task in the job step; work is still being performed when your task is unable to use the system while waiting for an event, such as an input operation, to occur.

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

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

CREATING THE TASK

3.5 A new task is created by issuing an ATTACH macro instruction. The task that is active when the ATTACH macro instruction is issued is the originating task; the newly created task is the subtask of the originating task. The subtask competes for control in the same manner as any other task in the system, on the basis of priority and the current ability to use the central processing unit. The address of the task control block for the subtask is returned in register 1.

3.6 If the ATTACH macro instruction is executed successfully, control is returned to the user with one of the following return codes in register 15:

Hexadecimal Code	Meaning
00	Indicates successful completion of the ATTACH request.
04	Indicates that the ATTACH macro instruction was issued in a STAE exit routine.
08	Indicates that sufficient main storage was not available to schedule the exit routine as specified by the STAI operand. The subtask has not been successfully created.
0C	Indicates that the exit routine or parameter list address specified in the STAI operand was invalid. The subtask has not been successfully created.
10	Indicates that storage for the STAI request is not available for the propagation of STAIs from the mother to the daughter task. The subtask has not been created.

3.7 The entry point in the load module to be given control when the subtask becomes active is specified in the same way as in a LINK macro instruction, that is, through the use of the EP, EPLOC, DE, and DCB operands. The use of these operands is discussed in the section titled "Program Management." Parameters can be passed to the subtask using the PARAM and VL operands, also described in "Program Management." Ownership of subpools is transferred or shared using the GSPV, GSPL, SHSPV, and SHSPL operands discussed in "Main Storage Management." The only additional operands are those dealing with the priority of the subtask, the operands that provide for communication between tasks, and the TASKLIB operand.

3.8 The TASKLIB operand is used to specify the address of an opened data control block (DCB) for a job library to be searched for the entry point name of the module being attached and for the subsequent modules accessed by the subtask. If the TASKLIB operand is not specified, the job library DCB address from the attaching task's TCB is propagated to the subtask.

3.9 Warning: All modules contained in the job library and task libraries for a job step should be uniquely named. If duplicate module names are contained in these libraries, the results are unpredictable.

TASK PRIORITY

3.10 In a system with MVT or MFT with subtasking, tasks compete for control on the basis of priority. When a task is created, it is assigned a priority that can later be revised upward or downward. It is also assigned a limit to its priority, a value equal to the highest priority the task can be assigned; this value is called the task's limit priority. The task's actual priority, the basis on which it competes for control, is called the task's dispatching priority.

3.11 A task can change its own dispatching priority but not its own limit priority. It can change both the dispatching and limit priorities of its subtasks, but cannot set the limit priority of a subtask higher than its own limit priority.

PRIORITY OF THE JOB STEP TASK IN MVT

3.12 The control program assigns limit and dispatching priorities to a job step task using input from parameters in one or more job control language statements. However, the limit and dispatching priorities of the job step task never exceed the dispatching priority of the Initiator. The limit priority of the job step task cannot be changed. The following list is in the order of importance assigned to the various methods of specifying priority. That is, force priority (item 1) is used, if specified. If force priority is not specified, DPRTY (item 2) is used, if specified, etc. The following list contains the calculations performed for user-specified priorities. This may be overriden by the control program if the calculation exceeds the LIMIT value specified in the EXEC statement in the Initiator procedure, or if the calculation exceeds the dispatching priority of the Initiator (see the paragraphs following the list).

1. The installation may specify that all job step tasks assigned to a particular job class are to have a specified initial dispatching priority called the force priority. See the MVT Guide for a description of the force priority operand. If a force priority is specified, the limit and dispatching priorities are calculated as follows:

   (value X 16) + 11

This overrides any specifications in items 2, 3, and 4.

2. In MVT, you may specify a dispatching priority for the job step using the DPRTY parameter on the EXEC statement for the job step. See the JCL Reference publication for a description of the DPRTY parameter. If the DPRTY parameter is specified, the priorities of the job step are calculated as follows:

   dispatching priority = (value1 X 16) + value2
   limit priority = (value1 X 16) + 15

   If value1 is not specified, a default of 0 is used.
   If value2 is not specified, a default of 11 is used.

   This overrides any specification in items 3 or 4.

3. You may specify the priority for all job step tasks using the PRTY parameter in the JOB statement. See the JCL Reference publication for a description of the PRTY parameter. If the PRTY parameter is specified, the priorities of the job step task are calculated as follows:

   dispatching priority = (value X 16) + 11
   limit priority = (value X 16) + 15

4. If no priority specification is made in items 1 through 3, the priority specified in the EXEC statement of the Reader/Interpreter procedure is used. See the MVT Guide for a description of the statements used in the Reader/Interpreter procedure. The priorities of the job step task are calculated as follows:

   dispatching priority = (value X 16) + 11
   limit priority = (value X 16) + 15

3.13 There is one additional parameter used to determine the priorities of a job step task. The LIMIT parameter in the EXEC statement of the Initiator procedure allows the installation to specify the maximum limit and dispatching priorities that can be assigned to a job step task. See the MVT Guide for a description of LIMIT parameter in the EXEC statement for the Initiator procedure. Both the maximum limit and dispatching priorities are calculated as follows:

(value X 16) + 11

3.14 If the LIMIT parameter is specified and one or both of the calculated priorities from items 1, 2, 3, or 4 is greater than the priority calculated using the LIMIT parameter, the priority calculated using the LIMIT parameter is assigned as the dispatching and/or limit priority that exceeded the LIMIT priority.

3.15 The calculated priorities, determined by items 1, 2, 3, or 4 above and adjusted to the LIMIT priority as necessary, are compared with the dispatching priority of the Initiator. The Initiator has a dispatching priority of 251 and a limit priority of 255 unless they were modified using the DPRTY parameter on the EXEC card for the Initiator procedure. If one or both of the calculated priorities of the job step task is greater than the dispatching priority of the Initiator, the dispatching priority of the Initiator is assigned as the limit and/or dispatching priority that exceeded the Initiator's dispatching priority.

PRIORITY OF THE JOB STEP TASK IN MFT

3.16 The limit and initial dispatching priorities of a job step task are always the same. These priorities are determined by the partition in

which the job step will execute. Figure 28 gives in column 2 ("Highest Dispatching") the limit and initial dispatching priority assigned to the job step task in each partition.

3.17 The job step task can lower its initial dispatching priority by use of the CHAP macro instruction. It can later use this macro instruction to revise its dispatching priority either upward or downward. Of course, it can never raise its dispatching priority above its initial dispatching (limit) priority.

PRIORITY OF SUBTASKS

3.18 When a subtask is created, the limit and dispatching priorities of the subtask are the same as the current limit and dispatching priorities of the originating task except when the subtask priorities are modified by using the LPMOD and DPMOD operands of the ATTACH macro instruction. The LPMOD operand specifies the number to be subtracted from the current limit priority of the originating task. The result of the subtraction is assigned as the limit priority of the new task. The DPMOD operand specifies the number to be added to the current dispatching priority of the originating task. The result of the addition is assigned as the dispatching priority of the new task, unless the number is greater than the limit priority. In that case, the limit priority value is used as the dispatching priority.

3.19 There are no absolute rules for assigning priorities to tasks and subtasks. Priorities should be assigned on the basis that tasks of higher priority will be given control when competing with tasks of lower priority. Tasks with a large number of input/output operations should be assigned a higher priority than tasks with little input/output because the tasks with much input/output will be in a wait condition for a greater amount of time. The lower priority tasks will be executed when the higher priority tasks are in a wait condition. When the input/output operation has completed, the higher priority tasks will get control so that the next operation can be started. In addition, if one or more subtasks must be completed before the originating task can proceed beyond a certain point, the subtasks that must be completed should be assigned a priority which will eliminate as much as possible a long wait time in the originating task.

3.20 Since tasks from other job steps are competing for control, the priority initially established for the subtasks may be too high or too low to properly process the job step. To correct this, the priorities of these tasks can be changed after the tasks have been created by using the CHAP macro instruction. The EXTRACT macro instruction, discussed later, can be used to determine the current dispatching and limit priorities of the current task and its subtasks. Note that each change of 16 in limit or dispatching priority is equivalent to a change of one in job priority.

3.21 The CHAP macro instruction changes the dispatching priority of the active task or one of its subtasks. By adding a positive or negative value, the dispatching priority of the active task or a subtask is changed. The dispatching priority of the active task can be made less than the dispatching priority of another task waiting for control. If this occurs, the waiting task would be given control after execution of the CHAP macro instruction.

3.22 The CHAP macro instruction can also be used to increase the limit priority of any of the active task's subtasks. The _active_ task cannot change its own limit priority. The dispatching priority of a subtask can be raised above its own limit priority, but not above the limit of the originating task. When the dispatching priority of a subtask is

raised above its own limit priority, the subtask's limit priority is automatically raised to equal its new dispatching priority.

TIME SLICING

3.23 Time slicing is an optional feature that allows tasks that are members of the "time-slice group" to share control of the CPU. When a member of the time-slice group has been active for a certain length of time, it is interrupted, and control is given to another member of the group. In this way, all member tasks are given equal slices of CPU time; no task can use the CPU to the exclusion of all others.

MFT SYSTEMS WITHOUT SUBTASKING

3.24 At system generation, your installation designates certain contiguous main storage partitions for time slicing. Your tasks (job steps) are members of the time-slice group if your job is assigned to one of these partitions. You control partition assignment through the CLASS parameter of your JOB statement.

MFT SYSTEMS WITH SUBTASKING

3.25 Any task or subtask is considered a member of a time-slicing group if its dispatching priority is within the range of the dispatching priorities assigned to partitions designated for time slicing.

3.26 During execution, a task or subtask can use the CHAP macro instruction to designate itself as a member of the time-slicing group if its limit priority is equal to or greater than the lowest dispatching priority of the time-slicing group. Also, a parent task can use the ATTACH or CHAP macro instructions to designate a subtask as a member of the time-slicing group if the limit priority of the parent task is equal to or greater than the lowest dispatching priority of the time-slicing group.

3.27 Each partition has a range of eleven dispatching priorities assigned to it. The range of dispatching priorities for a time-slicing group is from the highest dispatching priority of the highest priority partition within the group to the lowest dispatching priority for the lowest priority partition within the group. The highest and lowest dispatching priorities of a partition are given in Figure 28. The dispatching priorities indicated in the figure must be decremented by 1 for each of the following functions that are included in the system:

- System Log
- System Management Facility
- I/O Recovery Management Support

If Partitions 6 through 8 were assigned to the time-slicing group, any task or subtask whose dispatching priority fell within the range 185-153 would be a member of the time-slicing group. If the System Log and System Management Facility functions were included in the system, the range of time-slicing dispatching priorities would be 183-151.

Partition Number	Highest Dispatching	Lowest Dispatching
0	251	241
1	240	230
2	229	219
3	218	208
4	207	197
5	196	186
6	185	175
7	174	164
8	163	153
9	152	142
10	141	131
11	130	120
12	119	109
13	108	98
14	97	87
15	86	76
16	75	65
17	64	54
18	53	43
19	42	32
20	31	21
21	20	10
22	9	1
23-n	0	0

Figure 28. Determining partition dispatching priorities

MVT SYSTEMS

3.28 At system generation, your installation designates certain job priorities for time slicing. Your tasks are members of the time-slicing group if their dispatching priorities correspond to these job priorities. For example, if job priorities 8 and 9 are designated, tasks are members of the time-slice group when their dispatching priorities can be computed as follows:

For job priority 8,
Dispatching Priority = (8 x 16) + 11 = 139

For job priority 9,
Dispatching Priority = (9 x 16) + 11 = 155

In this example, tasks with priorities 139 and 155 are members of the time slice group. Note that time slicing applies only to ready tasks with the highest priority; a task with priority 155 would not be interrupted to give control to a task with priority 139.

3.29 Time slicing is important chiefly in real-time applications, but it affects the use of the ATTACH and CHAP macro instructions by all tasks in the system. These macro instructions determine task priorities, and therefore determine membership in the time slice group. In using these macro instructions, you must consider carefully the priorities for which time slicing is performed at your installation. Using the ATTACH and the CHAP macro instructions can affect dispatching priorities, as discussed above.

3.30 Consider again the example in which time slicing is performed for job priorities 8 and 9. If a job step task has an initial dispatching priority of 139, it is initially a member of the time-slice group. If it lowers its priority, it is no longer a member of the group; if it attaches a subtask, the subtask is a member only if it is assigned a dispatching priority of 139 (the limit priority of the job step task).

3.31 If another job step task is assigned an initial dispatching priority greater than 155, it is not initially a member of the time-slice group. However, it can create lower priority subtasks that are members of the time-slice group, and can itself become a member by lowering its own dispatching priority to 155 or 139. Note that careless use of the ATTACH and CHAP macro instructions could result in a task's becoming a member of the time-slice group when time slicing is not actually intended.

4.1 The task management information in this section is required only for establishing communications among tasks in the same job step, and therefore applies only to operating systems with MVT or with MFT with subtasking. The relationship of tasks in a job step is shown in Figure 29.

4.2 The horizontal lines in Figure 29 divide the tasks into various levels. These levels have no relation to task priorities; they serve only to separate originating tasks and subtasks. Tasks A, B, A1, A2, A2a, B1, and B1a are all subtasks of the job step task; Tasks A1, A2, and A2a are subtasks of Task A. Tasks A2a and B1a are the lowest level tasks in the job step. Although Task B1 is at the same level as Tasks A1 and A2, it is not considered a subtask of Task A.

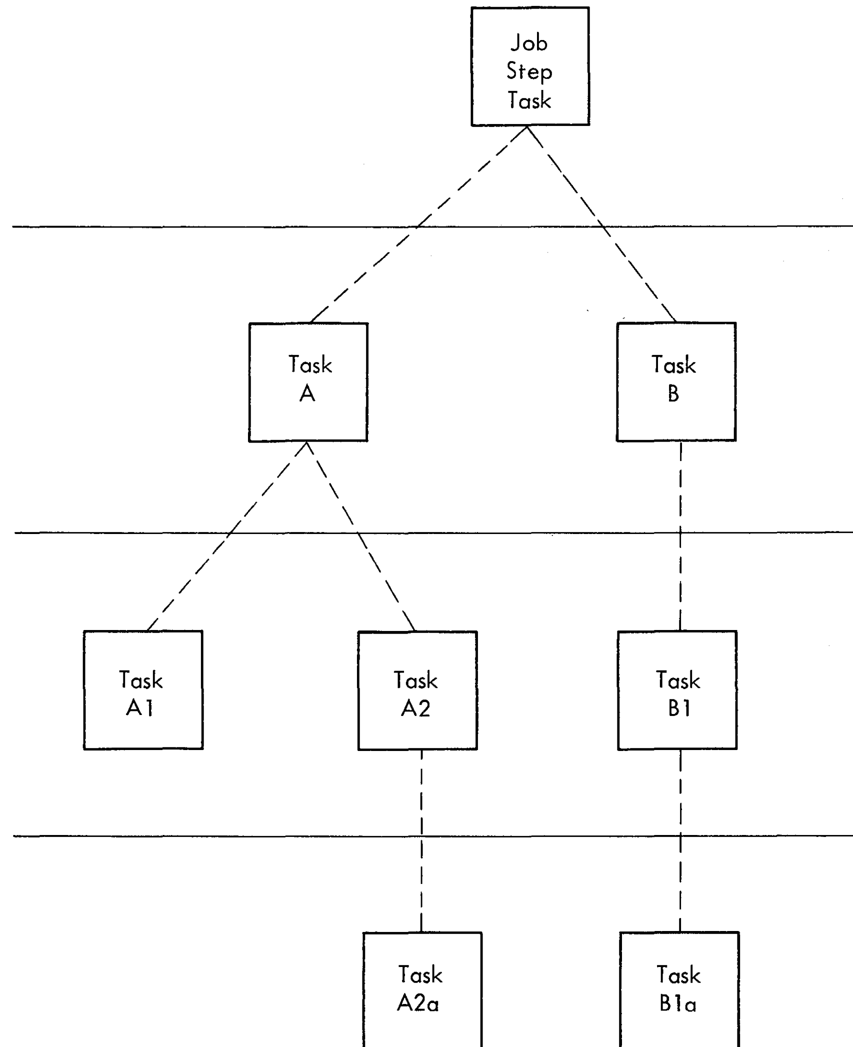


Figure 29. Task hierarchy

4.3 Task A is the originating task for both Tasks A1 and A2, and Task A2 is the originating task for Task A2a. A hierarchy of tasks exists within the job step. Therefore the job step task, Task A, and Task A2 are predecessors of Task A2a, while Task B has no direct relationship to Task A2a.

4.4 All of the tasks in the job step compete independently for control; if no constraints are provided, the tasks are performed and are terminated asynchronously. However, since each task is performing a portion of the same job step, you will usually require some communication and constraints between tasks, such as notification of the completion of subtasks. If termination of a predecessor task is attempted before all of the subtasks are complete, those subtasks and the predecessor task are abnormally terminated.

TASK AND SUBTASK COMMUNICATIONS

4.5 Two operands, the ECB and ETXR operands, are provided in the ATTACH macro instruction to assist in communication between a subtask and the originating task. These operands are used to indicate the normal or abnormal termination of a subtask to the originating task. If either the ECB or ETXR operands, or both, are coded in the ATTACH macro instruction, the task control block of the subtask is not removed from the system when the subtask is terminated. The originating task must remove the task control block from the system after termination of the subtask. This is accomplished by issuing a DETACH macro instruction. The task control blocks for all subtasks must be removed before the originating task can terminate normally.

4.6 The ETXR operand specifies the address of an end-of-task exit routine in the originating task to be given control when subtask being created is terminated. The end-of-task routine is given control asynchronously after the subtask has terminated, and must be in main storage when it is required. After the control program terminates the subtask, the end-of-task routine specified when the subtask was created is scheduled to be executed. The routine competes for control on the basis of the priority of the originating task, and can be given control even though the originating task is in the wait condition. When the end-of-task routine returns control to the control program, the originating task remains in the wait condition if the event control block has not been posted.

4.7 The end-of-task routine can issue an EXTRACT macro instruction specifying the task control block of the terminated subtask. The address of that task control block is contained in register 1 when the routine is given control. The EXTRACT macro instruction, discussed under the heading "Obtaining Information From the Task Control Block," can be used to obtain such information as floating-point register contents and completion code. Although the DETACH macro instruction does not have to be issued in the end-of-task routine, this is a good place for it.

4.8 The ECB operand specifies the address of an event control block (discussed under "Task Synchronization") which is posted by the control program when the subtask is terminated. After posting, the event control block contains the completion code specified for the subtask.

4.9 If neither the ECB nor ETXR operands are specified in the ATTACH macro instruction, the task control block for the subtask is removed from the system when the subtask is terminated. No DETACH macro instruction is required. Use of the task control block in a CHAP, EXTRACT, or DETACH macro instruction in this case is risky as is task termination; since the originating task is not notified of subtask termination, you may refer to a task control block which has been removed from the system, which would cause the active task to be abnormally terminated.

TASK SYNCHRONIZATION

4.10 Task synchronization requires some planning on your part to determine what portions of one task are dependent on the completions of portions of all other tasks. The POST macro instruction is used to signal completion of an event; the WAIT macro instruction is used to indicate that a task cannot proceed until one or more events have occurred.

4.11 The control block used with both the WAIT and POST macro instructions is the event control block. An event control block is a fullword on a fullword boundary and is shown in Figure 30.

4.12 An event control block is used when the ECB operand is coded in an ATTACH macro instruction. In this case the control program issues the POST macro instruction for the event (subtask termination). Either the return code in register 15 (if the task completed normally) or the completion code specified in the ABEND macro instruction (if the task was abnormally terminated) is placed in the event control block as shown in Figure 30. The originating task can issue a WAIT macro instruction specifying the event control block; the task will not regain control until after the event has taken place and the event control block is posted.

4.13 When an event control block is originally created, bits 0 and 1 must be set to zero. An event control block can be reused; if it is reused, bits 0 and 1 must be set to zero before either the WAIT or POST macro instruction can be issued. However, if the bits are set to zero before posting the ECB, any task waiting for that ECB to be posted will remain in the wait state. When a WAIT macro instruction is issued, bit 0 of the associated event control block is set to 1. When a POST macro instruction is issued, bit 1 of the associated event control block is set to 1, and bit 0 is set to 0.

4.14 A WAIT macro instruction can specify more than one event by specifying more than one event control block. Only one WAIT macro instruction can refer to an event control block at one time, however. If more than one event control block is specified in a WAIT macro instruction, the WAIT macro instruction can also specify that all or only some of the events must occur before the task is taken out of the wait condition. When a sufficient number of events have taken place (event control blocks have been posted) to satisfy the number of events indicated in the WAIT macro instruction, the task is taken out of the wait condition.

MANIPULATING TASK PROCESSING

4.15 In MVT systems, you can use the STATUS macro instruction to specify that a task is or is not to be dispatched by the system.

4.16 When you issue the STATUS macro instruction with the START or STOP operand, the system determines whether the specified subtask of the current task or all subtasks of the current task are to be modified. When you specify START, the stop/start count in the subtask TCB(s) is decreased and the nondispatchability flags are cleared. When you specify STOP, the stop/start count in the subtask TCB(s) is increased and the nondispatchability flags are set. The nondispatchability flags are set for a task only if the task has no system routine being executed for it. If a system routine is being executed for the task, the task is made nondispatchable when it no longer has a system routine being executed for it.

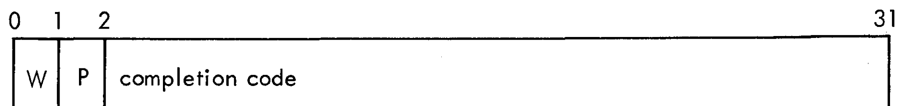


Figure 30. Event control block

PROGRAM MANAGEMENT SERVICES

5.1 The control program provides a set of optional services which are available to your program through the use of macro instructions. The following paragraphs discuss each of these services and the way to obtain them. The proper use of any of these services results in an improved and more efficient program; the misuse or overuse of the services wastes main storage and execution time.

ADDITIONAL ENTRY POINTS

5.2 Through the use of linkage editor facilities you can specify as many as 17 different names (a member name and 16 aliases) and associated entry points within a load module. It is only through the use of the member name or the aliases that a copy of the load module can be brought into main storage. Once a copy has been brought into main storage, however, additional entry points can be provided for the load module, subject to the following restrictions:

- The "identify" option must have been included in the operating system during system generation (standard in an operating system with MVT, optional with the other configurations of the operating system).
- The load module copy to which the entry point is to be added must be one of the following:
  - a copy which satisfied the requirements of a LOAD macro instruction issued during the same task, or
  - the copy of the load module most recently given control through the control program in performance of the same task.

5.3 The entry point is added through the use of the IDENTIFY macro instruction. An IDENTIFY macro instruction can be issued by any program in the job step, except by asynchronous exit routines established using other supervisor macro instructions. A further restriction exists for an operating system with MFT: an IDENTIFY macro instruction cannot be issued when the load module is given control at an entry point that was added by an IDENTIFY macro instruction.

5.4 When you use the IDENTIFY macro instruction, you specify the name to be used to identify the entry point, and the main storage address of the entry point in the copy of the load module. The address must be within a copy of a load module that meets the requirements listed above; if it is not, the entry point will not be added, and you will be given a return code of 0C (hexadecimal). The name can be any valid symbol of up to eight characters, and does not have to correspond to a name or symbol within the load module. The name must not be the same as any other name used to identify any load module available to the control program; duplicate names would cause errors. The control program checks the names of all load modules currently in the link pack area and the job pack area of the job step when you issue an IDENTIFY macro instruction, and provides a return code of 08 if a duplicate is found. You are responsible for not duplicating a member name or an alias in any of the libraries unintentionally.

5.5 The added entry point can be used only in an ATTACH macro instruction when you are using an operating system with MFT, and can be used in an ATTACH, LINK, LOAD, DELETE, or XCTL macro instruction in an operating

system with MVT. The added entry point can be used in the performance of any task in the job step; if the copy is in the link pack area, the entry point can be used in the performance of any task in the system.

5.6 The added entry point is available for as long as the copy is retained in main storage. Proper task synchronization is required when using an added entry point in the performance of a task which has not directly requested the associated copy of the load module; the load module may otherwise be deleted before the use is complete. The added entry point is treated by the control program as an entry point to a reenterable load module. You may use the IDENTIFY macro instruction to dynamically override nonreusable and only-loadable attributes. You must guard against improper changing of these attributes.

ENTRY POINT AND CALLING SEQUENCE IDENTIFIERS

5.7 An entry point identifier is a character string of up to 70 characters which can be specified in a SAVE macro instruction. The character string is created as part of the SAVE macro instruction expansion. The dump program uses the calling sequence identifier and the entry point identifier as shown in the Programmer's Guide to Debugging in the explanation of Save Area Trace.

5.8 A calling sequence identifier is a 16-bit binary number which can be specified in a CALL or a LINK macro instruction. When coded in a CALL or a LINK macro instruction, the calling sequence identifier is located in the two low-order bytes of the fullword at the return point address. The high-order two bytes of the fullword form a NOP instruction.

USING A SERIALLY REUSABLE RESOURCE

5.9 The example of a serially reusable resource already encountered was a load module that was designated serially reusable. In the discussion of the serially reusable load module it was emphasized that simultaneous uses of the load module must be prevented. This is true for any serially reusable resource when one or more of the users will modify the resource.

5.10 Consider a data area in main storage that is being used by programs associated with several tasks of a job step. Some of the users are only reading records in the data area; since they are not changing the records, their use of the data area can be simultaneous. Other users of the data area, however, are reading, updating, and replacing records in the data area. Each of these users must acquire, update, and replace records one at a time, not simultaneously. In addition, none of the users that are only reading the records wish to use a record that another user is updating, until after the record has been replaced. This illustrates the manner in which all serially reusable resources must be used.

5.11 For all of the uses of the serially reusable resource made during the performance of a single task, you must prevent incorrect use of the resource yourself. You must make sure that the logic of your program does not require the second use of the resource before completion of the first use. Be especially careful when using a serially reusable resource in an exit routine; since exit routines are given control asynchronously from the standpoint of your program logic, the exit routine could obtain a resource already in use by the main program. For the uses of the serially reusable resource required by more than one task, the ENQ macro instruction is provided to ensure use of the resource in a serial manner. The ENQ macro instruction cannot be used to prevent simultaneous use of the resource within a single task. It can only be used to test for simultaneous use within one task.

5.12 The ENQ macro instruction requests the control program to assign control of a resource to the active task. The control program determines the current status of the resource, and either grants the request by returning control to the active task or delays assignment of control by placing the active task in the wait condition. When the status of the resource changes so that control can be given to a waiting task, the task is taken out of the wait condition and placed in the ready condition. The use of the ENQ macro instruction is discussed in the following paragraphs.

NAMING THE RESOURCE

5.13 You represent the resource in the ENQ macro instruction by two names, known as the qname and the rname. These names may or may not have any relation to the actual name of the resource. The control program does not associate the name with the actual resource; it merely processes requests having the same qname and rname on a first-in, first-out basis. It is up to you to associate the names with the actual resource. It is up to all users of the resource to use qname and rname to represent the same resource. The control program treats requests having different qname and rname combinations as requests for different resources. Because the actual resource is not identified by the control program, it is possible to use the resource without issuing an ENQ macro instruction requesting it. If this happens, the control program cannot provide any protection.

5.14 If the resource is used only in the performance of tasks in your job step, you can assign the qname and rname combination. You should, in this case, code the STEP operand in the ENQ macro instructions that request the resource, indicating that the resource is used only in that job step. The control program will add the job step identifier to the rname so that no duplicate qname and rname combination will be used unintentionally in different job steps. If the resource is available to any job step in the system, the qname and rname combination must be agreed upon by all users and perhaps published. The SYSTEM operand should be coded in each ENQ macro instruction requesting one of these resources.

5.15 When selecting a qname for the resource, do not use SYS as the first three characters; qnames used by the control program start with SYS and you might accidentally duplicate one of these.

EXCLUSIVE AND SHARED REQUESTS

5.16 You can request exclusive or shared control of the resource for a task by coding either "E" or "S", respectively, in the ENQ macro instruction. If this use of the resource will result in modification of the resource, you must request exclusive control. If you are requesting use of a serially reusable load module and passing control yourself, as discussed previously, you must request exclusive control, since that program modifies itself during execution. If you are updating a record in a data area, you must request exclusive control. If you are only reading a record, and you will not change the record, you can request shared control. In order to protect any user of a serially reusable resource, all users must request exclusive or shared control on this basis. When a task is given control of a resource in response to an exclusive request, no other task will be given simultaneous control of the resource. When a task is given control of a resource in response to a shared request, control will be given to other tasks simultaneously only in response to other requests for shared control, never in response to requests for exclusive control. A request for shared control will protect against modification of the resource by another task only if the above rules are followed.

PROCESSING THE REQUEST

5.17 The control program essentially constructs a list for each qname and rname combination it receives in an ENQ macro instruction, and makes an entry in the list representing the task which is active when the ENQ macro instruction is issued. The entry is made in an existing list when the control program receives a request specifying a qname and rname combination for which a list exists; if no list exists for that qname and rname combination, a new list is built. The entry representing the task is placed on the list in the order the request is received by the control program; the priority of the task has no effect in this case. Control of the resource is allocated to a task based on two factors:

- The position on the list of the entry representing the task.
- The exclusive control or shared control requirements of the request which caused the entry to be added to the list.

5.18 The control program uses these two factors in determining whether control of a resource can be allocated to a task, as indicated below. Figure 31 shows the current status of a list built for a very popular qname and rname combination. The S or E next to the entry indicates that the request was for shared or exclusive control, respectively. The task represented by the first entry on the list is always given control of the resource, so the task represented by ENTRY 1 (Figure 31, Step 1) is assigned the resource. The request which established ENTRY 2 was for exclusive control, so the corresponding task is placed in the wait condition, along with the tasks represented by all the other entries in the list.

5.19 Eventually control of the resource is released for the task represented by ENTRY 1 and the entry is removed from the list. As shown in Figure 31, Step 2, ENTRY 2 is now first on the list, and the corresponding task is assigned control of the resource. Because the request which established ENTRY 2 was for exclusive control, the tasks represented by all the other entries in the list are kept in the wait condition.

5.20 Figure 31, Step 3 shows the status of the list after control of the resource is released for the task represented by ENTRY 2. Because ENTRY 3 is now at the top of the list, the task represented by ENTRY 3 is given control of the resource. ENTRY 3 indicated the resource could be shared, and, because ENTRY 4 also indicated the resource could be shared, ENTRY 4 is also given control of the resource. In this case, the task represented by ENTRY 5 will not be given control of the resource until control has been released for both the tasks represented by ENTRY 3 and ENTRY 4. The remainder of the list is processed in the same manner.

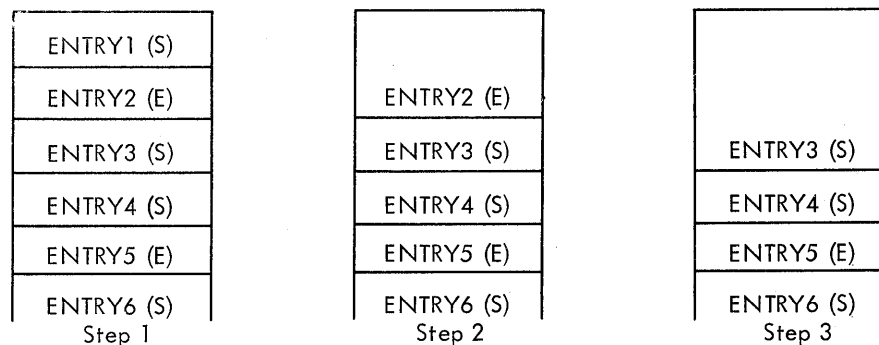


Figure 31. ENQ macro instruction processing

5.21 The following general rules are used by the control program:

- A task represented by the first entry in the list is always given control of the resource.
- If the request is for exclusive control, the task is not given control of the resource until the corresponding entry is the first entry in the list.
- If the request is for shared control, the task is given control either when the corresponding entry is first in the list or when all the entries before it in the list also indicate a shared request.
- If the request is for multiple resources, the task is given control when all of the entries for an exclusive request are first in the list and all of the entries for a shared request are either first in the list or are preceded only by entries for other shared requests.

PROPER USE OF ENQ AND DEQ

5.22 Proper use of the ENQ and DEQ macro instructions is required to avoid duplicate requests, to avoid tying up the resource, and to avoid interlocking the system. Guides to proper use are given in the following paragraphs.

DUPLICATE REQUESTS

5.23 A duplicate request occurs when an ENQ macro instruction is issued to request a resource if a task has already been assigned control of that resource or if a task is already waiting for that resource. If the second request results in a second entry on the list, the control program recognizes the contradiction and refuses to place the task in the ready condition (for the first request) and in the wait condition (for the second request) simultaneously. The second request results in abnormal termination of the task. You must plan the logic of your program to ensure that a second request for a resource is never issued until control of the resource is released for the first use. Again, be especially careful when using an ENQ macro instruction in an exit routine.

RELEASING CONTROL OF THE RESOURCE

5.24 The DEQ macro instruction is used to release control of a serially reusable resource assigned to a task through the use of an ENQ macro instruction. The task must be in control of the resource. Control of a resource cannot be released if the task does not have control. As you have seen, it is possible for many tasks to be placed in the wait condition while one task is assigned control of the resource. This may reduce the amount of work being done by the system. Issue a DEQ macro instruction as soon as possible to release control of the resource, so that other tasks can be performed. If you return to the control program at the end of processing for any task which is still assigned control of a resource, the resource is released automatically; however, in a system with MVT, the task is abnormally terminated.

CONDITIONAL AND UNCONDITIONAL REQUESTS

5.25 The normal use of the ENQ and DEQ macro instruction is to make unconditional requests. These are the only requests we have considered to this point. As you have seen, abnormal termination of the task occurs when two ENQ macro instructions are issued for the same resource in performance of the same task, without an intervening DEQ macro instruction. Abnormal termination also occurs if a DEQ macro instruction is issued in a task that has not been assigned control of the resource. Both of these abnormal termination conditions can be avoided either by more careful program design or through the use of the RET operand in the ENQ or DEQ macro instructions. The RET operand (RET=TEST, RET=USE, RET=CHNG and RET=HAVE for ENQ, RET=HAVE for DEQ) indicates a conditional request for control or release of control.

5.26 RET=TEST is used to test the status of the list for the corresponding qname and rname combination. An entry is never made in the list when RET=TEST is coded. Instead a return code is provided indicating the status of the list at the time the request was made. A return code of 8 indicates an entry for the same task already exists in the list. A return code of 4 indicates the task would have been placed in the wait condition if the request had been unconditional. A return code of 0 indicates the task would have been given immediate control of the resource if the request had been unconditional. RET=TEST is most useful when used to determine if the task has already been assigned control of the resource. It is less useful when used to determine the current status of the list and to take action based on that status. In the interval between the time the control program checks the status and the time the return codes are checked by your program and another ENQ macro instruction issued, another task could have been made active and the status of the list could have been changed.

5.27 RET=USE indicates to the control program that the active task is to be assigned control of the resource only if the resource is immediately available. A return code of 0 indicates that an entry has been made on the list and the task has been assigned control of the resource. A return code of 4 indicates that the task would have been placed in the wait condition if the request had been unconditional; no entry is made in the list. A return code of 8 indicates an entry for the same task already exists in the list. RET=USE can be best used when there is other processing that could be performed without using the resource. You would not want to wait for the resource as long as there was other work that you could do.

5.28 RET=CHNG indicates to the control program that the caller wishes to have exclusive control of the resource for which he is already enqueued. A return code of 0 indicates that the resource is immediately available and has been assigned to the exclusive control of the caller. Either the caller was already enqueued with the exclusive attribute, or the requested change from shared to exclusive was honored. A return code of 4 indicates that the requested change in attribute cannot be honored, because the caller is currently sharing the resource with another user. A return code of 8 indicates that the user was not enqueued for the resource when he requested the attribute change. Although this is an error condition, control is returned to the user.

5.29 RET=HAVE is used in both the ENQ and DEQ macro instructions. An ENQ macro instruction is processed as a normal request for control unless an entry for the same task already exists. A return code of 8 indicates an entry for the same task already exists in the list. A return code of 0 indicates that the task has been assigned control of the resource. A DEQ macro instruction is processed as a normal request to return control unless the task does not have control of the resource. A return code of 0 indicates that control of the resource has been released. A return

code of 8 indicates that the task does not have control of the resource (although the task may be in the wait condition because of a request for the resource). RET=HAVE can be used to good advantage in an exit routine to avoid abnormal termination.

5.30 Figures 59 and 60 in Section II summarize the return code meanings and formats respectively for the RET=HAVE operand of the DEQ macro instruction.

5.31 Figures 61 and 62 in Section II summarize the same items for the RET=TEST, RET=USE, RET=HAVE, and RET=CHNG operands in the ENQ macro instruction.

AVOIDING INTERLOCK

5.32 An interlock condition arises when two tasks are waiting for each other to complete, yet neither task can gain access to the resource it needs to complete processing. An example of an interlock situation is shown in Figure 32. Task A has exclusive access to resource M, and higher-priority Task B has exclusive access to resource N. Task B is placed in a wait condition when it requests exclusive access to resource M because M is accessible only by Task A. The interlock becomes complete when Task A requests exclusive access to resource N because N is accessible only by Task B. The same interlock would have developed if Task B issued a single request for multiple resources M and N prior to Task A's second request. However, the interlock would not have developed if both tasks had issued single requests for multiple resources. Other tasks requiring either of the resources are also in a wait condition because of the interlock, although in this case they have not contributed to the conditions which caused the interlock.

5.33 The above example involving two tasks and two resources is a simple example of an interlock situation. The example could be expanded to cover many tasks and many resources. It is imperative that interlock situations be avoided. The following procedures indicate some ways of preventing interlock situations:

5.34
- Do not request resources that are not immediately required. If you can use the serially reusable resources one at a time, you should request them one at a time, and release control for one before requesting control for the next.

5.35
- Request shared control as much as possible. If the entries in the lists shown in Figure 32 had indicated shared requests, there would have been no interlock. This does not mean you should indicate a request for shared control when you will modify the resource. It does mean that you should analyze your requirements for the resources carefully, and not make requests for exclusive control when requests for shared control would suffice.

5.36
- The ENQ macro instruction can be written to request control of more than one resource at a time. The requesting program is placed in a wait state until all of the requested resources are available. Those resources not being used by any other program immediately become exclusively available to the waiting program and are unavailable to any other programs that may request access to the resource. For example, instead of coding the two ENQ macro instructions shown in Figure 33, the one ENQ macro instruction shown in Figure 34 could be coded. If all requests were made in this manner, it would avoid the interlock shown in Figure 32. All of the requests for one task would be processed before any of the requests for the second task. The DEQ macro instruction should be written in the same manner to release the entire "set" of resources at once.

Task A	Task B
ENQ (M,A,E,8,SYSTEM)	
	ENQ (N,B,E,8,SYSTEM)
	ENQ (M,A,E,8,SYSTEM)
ENQ (N,B,E,8,SYSTEM)	

Figure 32. Interlock condition

```
ENQ   (NAME1ADD,NAME2ADD,E,8,SYSTEM)
ENQ   (NAME3ADD,NAME4ADD,E,10,SYSTEM)
```

Figure 33. Two requests for two resources

```
ENQ   (NAME1ADD,NAME2ADD,E,8,SYSTEM,NAME3ADD,NAME4ADD,E,10,SYSTEM)
```

Figure 34. One request for two resources

5.37 • If the use of one resource always depends on the use of a second resource, then the pair of resources can be defined as one resource in the ENQ and DEQ macro instructions. This procedure can be used for any number of resources that are always used in conjunction. There would be no protection of the resources if they are also requested independently, however. The request would always have to be for the set of resources.

5.38 • If there are many users of a group of resources and some of the users require control of a second resource while retaining control of the first resource, it is still possible to avoid interlocks. In this case the order in which control of the resources is requested should be the same for each user. For instance, if resources A, B and C are required in the performance of many tasks, the requests for control should always be made in the order of A, B and C. In this manner an interlock situation will not develop, since requests for resource A will always precede requests for resource B.

5.39 The above is not an exhaustive list of the procedures to be used to avoid an interlock condition. You could also make repeated requests for control specifying the RET=USE operand, which would prevent the task from being placed in the wait condition; if no interlock situation was developing, of course, this would be an unnecessary waste of execution time. The solution to the interlock problem in all cases requires the cooperation of all the users of the resources.

OBTAINING INFORMATION FROM THE TASK CONTROL BLOCK

5.40 Most of the information available from the task control block is useful primarily in task management. The following paragraphs discuss the information available and how to obtain it. How you use the information provided depends on the application of your program.

5.41 The EXTRACT macro instruction is used to obtain information from the task control block (TCB), the command scheduler control block (CSCB), and the interruption request block (IRB). The full power of the EXTRACT macro instruction is available (and needed) only in an operating system with MVT or MFT with subtasking. However, a limited amount of information can be obtained through the use of the EXTRACT macro instruction with the other configuration of the operating system.

5.42 Information can be obtained from the TCB, CSCB, and IRB for the active task or any of its subtasks. The following information can be requested:

5.43 TCB

- The address of the general and floating point register save areas. These are the save areas used by the control program when the task is not active.
- The limit and dispatching priorities of the specified task.
- The completion code if the task has been terminated. If the specified task has not been terminated, the completion code value is set to zero.
- The address of the time sharing flags field (TCBTSFLG) and the protected storage control block (PSCB) from the job step control block (JSCB). This information can be obtained only when using EXTRACT in an operating system with the time sharing option (TSO).

5.44 CSCB

- The addresses of the task input/output table (TIOT) and of the command scheduler communications list in the command scheduler control block (CSCB). (The CSCB is in the system queue area.) These addresses are the only information provided in response to an EXTRACT macro instruction when using an operating system with MFT without subtasking.

5.45 IRB

- The address of the end-of-task exit routine to be given control after the specified task is terminated.

5.46 You must provide an area into which the control program places the information you request. If you request the fields GRS, FRS, AETX, PRI, CMC, and TIOT by coding FIELDS=ALL, the area must be seven fullwords long. If you request only a portion of the information, the area must be one fullword in length for each item of information you request. In a system with the time sharing option (TSO) you can also request the fields TSO, PSB, and TJID. If you request information other than the address of the task input/output table when you are using an operating system with MFT without subtasking, each additional item of information requested will result in the corresponding fullword in the answer area being set to zero.

TIMING SERVICES

5.47 The timing services available depend on options selected when the operating system was generated. These options are the time option, which provides the ability to request the date and time of day, and the interval option, which includes the time option functions and also provides the ability to set, test, and cancel intervals of time. The interval option is standard in an operating system with MVT; either option can be selected with the other configurations of the operating system. If neither of these options was selected, the date is the only

timing service provided. In the Model 65 Multiprocessing system, timing services must only be obtained through the use of the supervisor macro instructions: STIMER, TIME, TTIMER. Direct reference to the interval timer location in a multiprocessing system may produce unpredictable results.

DATE AND TIME OF DAY

5.48 The operator is responsible for initially supplying the correct date and time of day information, based on a 24-hour clock, for control program use. The time of day information is updated every 16.7 milliseconds for 60 cycle-per-second line frequency, or every 20 milliseconds for 50 cycle-per-second line frequency. You request the date and time of day information using the TIME macro instruction. The control program returns the date in register 1 and the time of day in register 0.

5.49 The date is returned in register 1 as packed decimal digits of the form 00yydddc, where yy are the last two digits of the year and ddd is the day of the year. C is the sign character hexadecimal F, which allows the year and day information to be unpacked directly for printing. One procedure used to request the day of the year is shown in Figure 35.

5.50 The time of day is returned in register 0 in the form specified in the TIME macro instruction. The time of day is returned as an unsigned 32-bit binary number that specifies the elapsed number of either hundredths of a second, if BIN is coded, or timer units, if TU is coded. (A timer unit is equal to 26.04166 micro-seconds.) If DEC is coded or the operand is omitted, the time of day is returned as packed decimal digits of the form HHMMSSth (hours, minutes, seconds, tenths of a second, and hundredths of a second). The packed decimal digits can be unpacked by changing the "h" value to a zone sign and using an UNPK instruction or by inserting zones between each decimal digit. If both the time and interval options have not been selected, the operand is ignored and the content of register 0 is set to zero.

TIMING SERVICES ON THE IBM SYSTEM/370

5.51 In an MFT or MVT system generated for System/370 all references to time of day and date use the time-of-day (TOD) clock. The TOD clock, a feature of System/370, is a 64-bit binary counter. (For more information about the TOD clock, see IBM System/370 Principles of Operation.) Bit 51 of the counter is equivalent to one microsecond.

5.52 The TOD clock is incremented continuously while the power is on; the clock is not affected by the system stop conditions that affect the interval timer in location 80. The operator normally sets the clock only after an interruption of CPU power has caused the clock to stop and restoration of power has restarted it. The operator sets the clock using the SET command with the DATE and CLOCK parameters.

```
         ...
         TIME                      ,   Request date
         ST    1,ANS                   Store packed date
         UNPK  DOUBLE,ANS              Unpack date for printing
         ...
ANS      DS    F                       Fullword for packed date
DOUBLE   DS    D                       Double word for unpacked date
```

Figure 35. Day of year processing

DATE AND TIME OF DAY

5.53 If you use the TIME macro instruction with the BIN, TU, and DEC operands, the date is returned in register 1 and the time of day is returned in regsiter 0. With the MIC,address operand, the time of day is returned as an unsigned 64-bit binary number in the area specified by "address." The time of day is returned with bit 51 equivalent to one microsecond. With the MIC,address operand, register 0 is set to zero.

INTERVAL TIMING

5.54 A time interval can be established for any task in the job step through the use of the STIMER macro instruction, and the time remaining in the interval can be tested and canceled through the use of the TTIMER macro instruction. When you are using an operating system with MFT without subtasking, only one time interval can be in effect at any one time during the job step. With an operating system with MVT or MFT with subtasking, each task in the job step can have an active time interval.

5.55 The time interval can be established by any one of the following four methods.

- BINTVL - requires an unsigned 32-bit binary number, the low order bit having a value of 0.01 second.
- TUINTVL - requires an unsigned 32-bit binary number, the low order bit having a value of 26.04166 microseconds (1 timer unit).
- DINTVL - requires an 8-byte field containing unpacked decimal digits of the form HHMMSSth (hours, minutes, seconds, tenths and hundredths of a second, based on a 24-hour clock).
- TOD - requires an 8-byte field similar to the field required for DINTVL. The control program interprets the time specified as the time of day at which the interval is to expire.

5.56 When you test the time remaining in the interval, the time remaining is returned as a 32-bit unsigned binary number in register 0, the low order bit having a value of 26.04166 microseconds. If the interval has already expired, the content of register 0 is set to zero.

5.57 When you request a time interval, you also specify the manner in which the interval is to be decremented, through the use of the TASK, REAL, or WAIT parameter of the STIMER macro instruction. REAL and WAIT both indicate that the interval is to be decremented continuously whether the associated task is active or not. TASK indicates that the interval is to be decremented only when the associated task is active. If REAL or TASK is coded, the task continues to compete with the other ready tasks for control; if WAIT is coded, the task is placed in the wait condition until the interval expires, at which time the task is placed in the ready condition.

5.58 When TASK or REAL is designated, the address of a timer completion exit routine can be specified. This is the first routine to be given control when the associated task is made active after the completion of the time interval. (If the address of the exit routine is not specified, there is no notification of the completion of the time interval.) The exit routine must be in main storage when required, and must save and restore registers and return control to the address in register 14. After control is returned to the control program, control is passed to the next instruction in the main program.

```
        ...
        STIMER TASK,FIXUP,BINTVL=TIME   Set time interval
LOOP    ...
        TM     TIMEXP,X'01' Test if fixup routine entered
        BC     1,NG         Go out of loop if time interval expired
        BXLE   12,6,LOOP    If processing not complete, repeat loop
        TTIMER CANCEL       If loop completes, cancel remaining time
        ...
NG      ...
        ...
        USING  FIXUP,15     Provide addressability
FIXUP   SAVE   (14,12)      Save registers
        OI     TIMEXP,X'01' Time interval expired, set switch in loop
        ...
        RETURN (14,12)      Restore registers
        ...
TIME    DC     X'00000200'  Time is 5.12 seconds
TIMEXP  DC     X'00'        Timer switch
```

Figure 36. Interval timing

5.59 Figure 36 shows the use of a time interval when testing a new loop in a program. The STIMER macro instruction sets a time interval of 5.12 seconds, to be decremented only when the task is active, and provides the address of a routine called FIXUP to be given control when the time interval expires. The loop is controlled by a BXLE instruction.

5.60 The loop continues as long as the value in register 12 is less than or equal to the value in register 6. If the loop completes, the TTIMER macro instruction causes any time remaining in the interval to be canceled; the exit routine is not given control. If, however, the loop is still in effect when the time interval expires, control is given to the exit routine FIXUP. The exit routine saves registers and turns on the switch tested in the loop. The FIXUP routine could also print out a message indicating that the loop did not complete successfully. Registers are restored and control is returned to the control program. The control program returns control to the main program and processing continues. When the switch is tested this time, the branch is taken out of the loop.

5.61 If issued by a timer completion exit routine, a STIMER macro instruction acts as a NOP instruction only for MFT. An exit routine therefore cannot be used to set a new time interval for MFT.

5.62 If issued by a timer completion exit routine, a STIMER macro instruction is honored for MVT. However, the STIMER issued from the exit routine should not specify that same exit routine. If it does specify the same exit routine, an infinite loop may occur.

5.63 The accuracy of a time interval is affected by two factors: the resolution of the timer and the "competition" of other tasks for control. The resolution of the timer (the time between successive updating of the timer) is 16.7 milliseconds for 60 cycle per second line frequency. An attempt to measure an interval of less than 16.7 milliseconds or an attempt to time to an accuracy of greater than 16.7 milliseconds can lead to erroneous results.

5.64 The priorities of other tasks in the system may also affect the accuracy of the time interval measurement. If you code REAL or WAIT, the interval is decremented continuously and may expire when the task is not active. (This is certain to happen when WAIT is coded.) After the time interval expires, assuming the task is not in the wait condition for any other reason, the task is placed in the ready condition and then competes for control with the other tasks in the system that are also in

the ready condition. The additional time required before the task becomes active will then depend on the relative dispatching priority of the task.

WRITING TO ONE OR MORE OPERATOR CONSOLES

5.65 The WTO and the WTOR macro instructions allow you to write messages to the operator's console and/or to the system message class data set, depending on the routing code specified. The WTOR macro instruction also allows you to request a reply from the operator. When Multiple Console Support (MCS) is included in the system, messages can be sent to (and replies can be received from) as many as 32 operator consoles.

5.66 There are two basic forms of the WTO macro instruction: the single-line form, and the multiple-line form. To use the single-line form, code the single-line message within apostrophes. The message that the operator receives does not contain these apostrophes. The message can include any character that is valid in a character (C-type) DC instruction, except the new-line control character (hexadecimal value 15). It is assembled as a variable-length record, which is written automatically; you do not have to provide a data control block.

5.67 To use the multiple-line form of the macro instruction, code the text of each line within apostrophes followed by a line type indicator. Enclose both of these items in one set of parentheses. Up to ten contiguous lines of information may be passed to the operator's console by a problem program.

5.68 The following should be considered when issuing multiple-line WTO messages:

- Tasks issuing multiple-line WTO messages will not be rolled out until the multiple-line message is ended.
- Multiple-line WTO messages are not passed to the user-written WTO exit routine.
- When a console switch takes place, unended multiple-line WTO messages and multiple-line WTO messages in the process of being written to the original console are not moved to the new console.
- When the system hard copy log is an active operator's console, only the hard copy versions of multiple-line messages are written to the console.
- An active operator's console should be used as the hard copy log only in an emergency.

5.69 See the macro instructions section for an explanation of the parameters in the multiple-line form of the WTO macro instruction.

5.70 Routing of the message (in a system with the MCS option) is performed using the routing codes specified in the WTO macro instruction. At system generation, each operator's console in the system is assigned routing codes which correspond to the functions that the installation wants that console to perform. When any of the routing codes assigned to a message match any of the routing codes assigned to a console, the message is sent to that console. For more information about routing codes, refer to Appendix A.

5.71 Disposition of the message (in a system with the MCS option) is indicated through the descriptor codes specified in the WTO macro instruction. Descriptor codes functionally classify WTO messages so that they may be properly presented on, and deleted from, display type devices.

Each WTO macro instruction should contain one descriptor code. The descriptor code is not printed or displayed as part of the message text. If a descriptor code of one or two is coded into the WTO macro instruction, an asterisk (*) is inserted as the first character of the message. The asterisk informs the operator that he is required to take some immediate action. If a descriptor code other than one or two is coded, a blank is inserted as the first character, indicating that no immediate action is needed. For more information about descriptor codes, refer to Appendix A.

5.72 A sample WTO macro instruction is shown in Figure 37. The routing code (ROUTCDE) and descriptor code (DESC) keyword parameters are ignored if the operating system does not have the MCS option.

5.73 To use the WTOR macro instruction, code the message exactly as designated in the single-line format of the WTO macro instruction (the WTOR macro instruction cannot be used to pass multiple-line messages). When the message is written, the control program adds a two-character message identifier before the message to associate the reply with the message. The control program also inserts an asterisk as the first character of all WTOR messages, thereby informing the operator that immediate action is required. You must, however, indicate the operator response desired. In addition, you must supply the address of the area in which the control program is to place the reply, and you must indicate the length of the reply. You also supply the address of an event control block which the control program will post after the reply has been placed, left-adjusted, in your designated area. (The use of the event control block is discussed under the heading "Task Management.")

5.74 A sample WTOR macro instruction is shown in Figure 38. The routing code and descriptor code values are ignored if the operating system does not have the MCS option. The reply is not necessarily available at the address you specified until a WAIT macro instruction has been issued.

5.75 When a WTOR macro instruction is issued to more than one functional area (where the WTOR has more than one routing code), any console within those areas has the authority to reply. The first reply received by the operating system is returned to the issuer of the WTOR, providing the syntax of the reply is correct. If the syntax of the reply is not

```
Single-line WTO 'BREAKOFF POINT REACHED.  TRACKING COMPLETE',         C
format          ROUTCDE=14,DESC=7

Multiple-   WTO ('SUBROUTINES CALLED',C),('ROUTINE TIMES CALLED',L), C
line format     ('SUBQUER             ',D),('ENQUER     ',D),          C
(List form)     ('WRITER              ',D),('DQUER      ',DE),         C
                ROUTCDE=(2,14),DESC=(7,8),MF=I
```

Figure 37. Writing to the operator

```
        ...
        XC    ECBAD,ECBAD         Clear ECB
        WTOR  'STANDARD OPERATING CONDITIONS?  REPLY YES OR NO',     C
              REPLY,3,ECBAD,ROUTCDE=(1,15),DESC=7
        WAIT  ECB=ECBAD
        ...
 ECBAD  DC    F'0'                Event control block
 REPLY  DC    C'bbb'              Answer area
```

Figure 38. Writing to the operator with a reply

correct, another reply is accepted. The WTOR is satisfied when the operating system moves the reply into the issuer's reply area and posts the event control block as completed. Each console that received the original WTOR will also receive the accepted reply. The master console operator may answer any WTOR, even if he did not receive the original message.

WRITING TO THE PROGRAMMER

5.76 The WTO macro instruction (single-line format only) and the WTOR macro instruction allow you to write messages to the programmer, as well as to the operator.

5.77 At system generation (SYSGEN) time, your installation determines how many 176-byte system message blocks (SMBs) to allow. You can override this number at initial program load (IPL) time; however, the number of SMBs allowed must range from 1 to 20.

5.78 When you submit your job, you can specify the message output class for your messages by using the MSGCLASS parameter of the JOB statement. (For a description of the MSGCLASS parameter, refer to the Job Control Language Reference manual.) All WTO and WTOR messages within the number of SMBs allowed per job will appear in the designated message output class. When you exceed the number of allowable SMBs, no subsequent user-issued WTP messages will appear in the message output class.

5.79 To write a message to the programmer, you must specify ROUTCDE=11 in the WTO or the WTOR macro instruction. If you use routing code 11 alone or together with other routing codes, the message goes to the message output class, as described above. The message can also go to the console(s) in the situations described by Figure 39.

If you specify a routing code of 11 (ROUTCDE=11)		
In this macro instruction:	In a system:	Your message goes to the:
WTO	With MCS	Message output class Consoles designated to receive messages with ROUTCDE=11
WTO	Without MCS	Message output class
WTOR	With MCS	Message output class Master console
WTOR	Without MCS	Message output class Master Console
If, in addition to routing code 11, you specify the appropriate routing code(s) in either a WTO or a WTOR macro instruction with or without MCS, the message appears on the console(s) designated to receive the routing code(s). In addition, the message appears in the same places as it does when you specify only routing code 11 (as shown above), with one exception. For WTOR with MCS, the message goes to the master console only if you specify that console's routing code.		

Figure 39. Using WTO and WTOR to write messages to the programmer

WRITING TO THE HARD COPY LOG

5.80 When using an operating system that has the Multiple Console Support (MCS) option, you can record information on the hard copy log. Since the MCS option allows more than one console in a system, an installation might find it helpful to be able to record all the messages issued by and to a system. The hard copy log provides a place to collect these messages, and therefore allows an installation to review system activity by reviewing message activity.

5.81 Since the hard copy log is optional, you should know whether your system was generated with it. The hard copy log is either an operator's console with output capability or the system log.

5.82 To record information on the hard copy log, you use the WTO or WTOR macro instruction. Your installation must have decided which system functions are to be logged and assigned appropriate routing codes to the hard copy log. The routing codes that you assign to your WTO or WTOR macro instruction are compared to the routing codes assigned to the log. If one or more codes match, the message is entered in the log. This means you do not have to issue a WTL macro instruction to record system and problem program information when the same information is going to the operator. You must, however, know which system functions the log is recording and assign an appropriate routing code to your WTO or WTOR macro instruction.

5.83 For each entry in the hard copy log, both the time when the message is received by the system and the routing codes for the message are appended to the beginning of the message text. Recording the time that the message was received, a procedure called time stamping, allows you to obtain a chronological record of system activity. For a system that does not have the timer option, the space for time stamping is filled with zeros.

5.84 Whether the hard copy log is the operator's console or the system log, the hard copy log information cannot be confused with other information. This is because the hard copy log entries are prefixed with the time stamp and the routing codes.

WRITING TO THE SYSTEM LOG

5.85 Operating systems with MFT, MVT, or Model 65 Multiprocessing provide a system log as an optional feature. The system log consists of two SYSOUT data sets on which the communication between the operator and the system is recorded. You can use the system log by coding the information that you wish to log in the "text" operand of the WTL macro instruction.

5.86 The data set receiving data from the system, user programs, and/or operators is the primary data set. The data set being written, or waiting to be written, to a system output device is the alternate data set. The primary data set, the one that is currently open and receiving input, is logically connected to two buffers. The operating system fills one buffer and writes it to the primary data set while filling the other buffer. The alternate data set has been logically disconnected from the buffers because it has been filled and must wait to be written to a system output device. After being written to a system output device, the alternate data set can be used again to receive input. When receiving input, the alternate data set becomes the primary data set.

5.87 When the WTL macro instruction is executed, the system places your text in one of the buffers and, when the buffer is full, writes the buffer onto the system log primary data set. The system writes the text of your WTL macro instruction on the master console instead of on the system log if one of the following two conditions exists:

- The system log is nct supported.
- The system log is supported, but the system log data sets are temporarily inactive because both are full and waiting to be written.

Your installation probably has an operator procedure to follow for both of the above conditions.

5.88 Although when using the WTL macro instruction you code the message within apostrophes, the written message does not contain the apostrophes. The message can include any character that is valid for the WTL macro instruction and is assembled and written the same way as the WTO macro instruction. MCS routing codes and descriptor codes are not assigned since they are not needed by the WTL macro instruction.

MESSAGE DELETION

5.89 If your system is using an operator console with a cathode ray tube (CRT) display screen, unnecessary messages can be deleted from the screen by the programmer.

5.90 The operating system assigns a message identification number to each WTO and WTOR message, and returns the message to the program in register 1. The DOM macro instruction uses the identification number to indicate which message is to be deleted. The message identification number must not be confused with the reply identification number that is assigned to WTOR replies.

OPERATOR COMMUNICATION WITH A PROBLEM PROGRAM

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

5.92 An EXTRACT macro instruction is issued to obtain a pointer to the communications ECB, which is posted when a STOP or a MODIFY command is issued, and a pointer to the first command input buffer (CIB) on the CIB chain for the task:

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

EXTRACT will return the following:

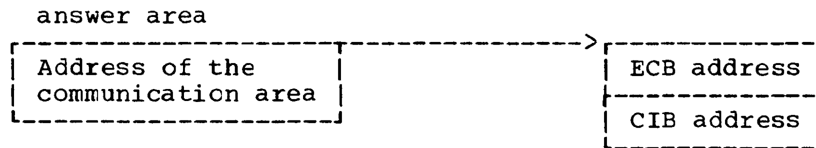
5.93 The CIB contains the information specified on the STOP or the MODIFY command, as shown in Figure 40. If the job was started from the console, the CIB pointed to when the EXTRACT macro instruction is issued will be the START CIB. If the job was not started from the console, the address of the first CIB will be zero. If the address of the START CIB is present, the QEDIT macro instruction should be used to free this CIB after any parameters passed in the START command have been examined:

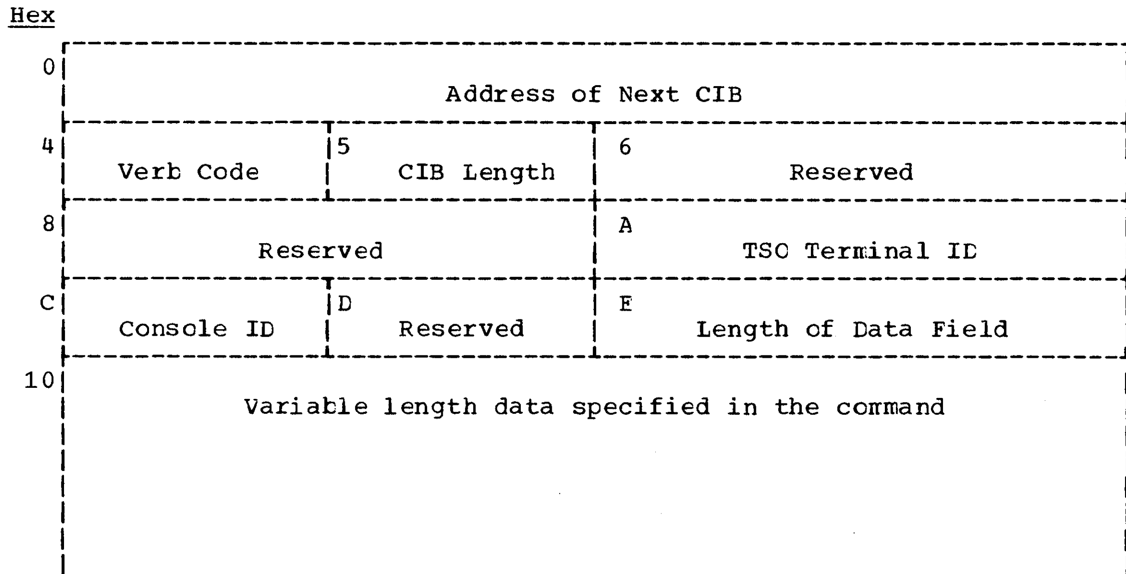
```
QEDIT ORIGIN=address of pointed to CIB,BLOCK=address of CIB
```

The CIB counter should then be set to allow CIBs to be chained and MODIFY commands accepted for the job. This is also accomplished by using the QEDIT macro instruction (the QEDIT macro instruction is described in the _MFT_ and _MVT Guides_):

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

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





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

Figure 40. Command input buffer contents

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

5.95 The communications ECB will be cleared when the last CIB on the chain is freed. Care should be taken if multiple subtasks are examining these fields. Any CIBs not freed by the task will be unchained by the system when the task is terminated. The area addressed by the pointer obtained by the EXTRACT macro instruction, the communications ECB, and all CIBs are in protected main storage and may not be altered.

GENERALIZED TRACE FACILITY (GTF) INTERFACE

5.96 One of the capabilities of the Generalized Trace Facility (GTF) is the recording of data originated by application programs. The interface between the application programs and GTF is the GTRACE macro instruction. For a complete discussion of GTF, see the Service Aids publication.

5.97 GTRACE allows from 1 to 256 bytes of data to be entered in a GTF buffer and recorded. When the GTRACE macro instruction is executed, GTF must be active and conditioned to receive application data and to record this data on an external device; otherwise the data will not be accepted. The data to be traced must be in your partition or region.

Return codes are used to indicate the result of the operation. The GTRACE macro instruction is fully described in the macro instructions section.

5.98 Recorded data is processed by the edit function of the IMDPRDMP service aid. If you want more than a hexadecimal dump of the records, you may prepare formatting routines for use with the IMDPRDMP edit function. Association between your recorded data and the formatting routine by which it is processed is established by entering a format identifier in the GTRACE macro instruction. This identifier defines the formatting routine that is to process the record. The <u>Service Aids</u> publication contains a complete discussion of IMDPRDMP.

5.99 To use the GTRACE macro instruction, specify the address and the number of bytes of data to be entered, along with an event identifier. A unique event identifier may be specified each time the GTRACE macro instruction is coded. This identifier may be used, for example, in output record identification. The optional FID= parameter indicates the formatting routine to be used by IMDPRDMP in processing the record. In Figure 41, 200 bytes of data, beginning at the address of AREA, are to be recorded with an event identifier of 37. When the record is edited and printed by the IMDPRDMP service aid, a routine designated by suffixing the FID value (converted to Hexadecimal) to the characters IMDUSR will be loaded to process the record. In Figure 41, IMDUSR28 is designated.

```
GTRACE DATA=AREA,LNG=200,ID=37,FID=40
```

Figure 41. Specifying the GTRACE macro instruction

PROGRAM INTERRUPTION PROCESSING

5.100 Unusual conditions encountered in a program cause a program interruption. These conditions include incorrect operands and operand specifications, as well as exceptional results, and are known generally as <u>program exceptions</u>. For certain exceptions (fixed-point and decimal overflow, exponent underflow and significance), interruptions can be disabled by setting the corresponding bits in the program status word to zero.

5.101 When a task becomes active for the first time, all program interruptions that can be disabled are disabled, and a standard control program exit routine, included when the system was generated, is provided. This control program exit routine is given control when any program interruptions occur, and issues an ABEND macro instruction specifying task abnormal termination and requesting a dump. By issuing the SPIE macro instruction, you can specify your own exit routine to be given control for one or more types of program exception. The macro instruction specifies the address of the exit routine to be given control when specified program exceptions occur. If the SPIE macro instruction specifies an exception for which the interruption has been disabled, the control program enables the interruption when the macro instruction is issued.

5.102 The SPIE macro instruction can be issued by any program being executed in performance of the task. When the task is active, your exit routine receives control for all interruptions resulting from exceptions specified in the SPIE macro instruction. For other program interruptions, control is given to the control program exit routine. Each suc-

ceeding SPIE macro instruction completely overrides specifications in the previous macro instruction.

PROGRAM INTERRUPTION CONTROL AREA

5.103 The expansion of each standard or list form SPIE macro instruction contains a control program parameter list called the program interruption control area (PICA). The PICA and another control program area called the program interruption element (PIE) contain the information that enables the control program to intercept user-specified program interruptions. Together, the PIE and the PICA associated with it are called the "SPIE environment." (The PIE is described later in this section.) The PICA, as shown in Figure 42, contains the new program mask for the interruption types that can be disabled, the address of the exit routine to be given control when one of the specified interruptions occurs, and a code for interruption types (exceptions) specified in the SPIE macro instruction.

5.104 The control program maintains a pointer (in the PIE) to the PICA referred to by the last SPIE macro instruction executed. This PICA may have been created by the last SPIE (standard or list form) or may have been created previously and referred to by the last SPIE (execute form). Each program that issues a SPIE macro instruction, before returning control to the calling program or passing control to another program by issuing an XCTL macro instruction, must cause the control program to adjust the SPIE environment to the condition that existed or to eliminate the SPIE environment if one did not exist on entry to the program. When the standard form or execute form of the SPIE macro instruction is issued, the control program returns the address of the previous PICA in register 1. If no SPIE environment existed when the program was entered, the control program returns zeros in register 1.

5.105 The effect of the last SPIE macro instruction is canceled by issuing a SPIE macro instruction with no operands. This action does not reestablish the effect of the previous SPIE; it does create a new PICA that contains zeros, thus indicating that no user exit routine is to process interruptions. Any previous SPIE environment may be reestablished, regardless of the number or type of subsequent SPIE macro instructions issued, by using the execute form of the SPIE macro instruction specifying the appropriate value that had been returned in register 1 by the control program. If a PICA address is specified (as opposed to zeros), the PICA must be still valid (not overlaid). The SPIE environment will be eliminated by specifying zeros as the PICA address.

5.106 Figure 43 shows how to restore a previous PICA. The first SPIE macro instruction designates an exit routine called FIXUP that is to be given control if fixed-point overflow occurs. The address returned in register 1 is stored in the fullword called HOLD. At the end of the program, the execute form of the SPIE macro instruction is used to restore the previous PICA.

PROGRAM INTERRUPTION ELEMENT

5.107 When the first SPIE macro instruction is executed in performance of a task, the control program creates a 32-byte program interruption element (PIE) in the main storage area assigned to the job step (subpool 0 in MVT). Because the PIE is freed when the SPIE environment is eliminated (by specifying a PICA address of zero in the execute form of a SPIE macro instruction), a PIE will also be created whenever a SPIE macro instruction is issued and no PIE exists. The format of the PIE is shown in Figure 44.

5.108 The PICA address in the program interruption element is the address of the program interruption control area used in the last execution of a SPIE macro instruction for the task. When control is passed to the routine indicated in the PICA, the old program status word contains the interruption code in bits 16-31; these bits can be tested to determine the cause of the program interruption. The contents of register 14, 15, 0, 1, and 2 at the time of the interruption are stored by the control program as indicated.

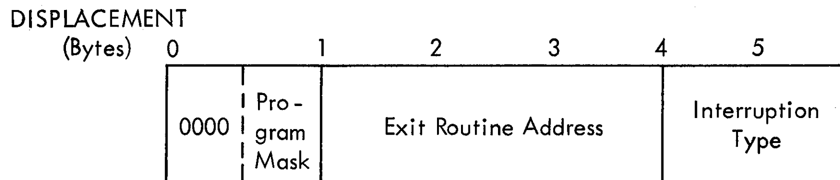


Figure 42. Program Interruption Control Area

```
        SPIE  END,(4)      Provide exit routine for protection
        ...
        SPIE  FIXUP,(8)    Provide exit routine for fixed-point overflow
        ST    1,HOLD       Save address returned in register 1
        ...
        L     5,HOLD       Reload returned address of PICA for first SPIE
        SPIE  MF=(E,(5))   Use execute form and old PICA address
        ...
  HOLD  DC    F'0'
```

Figure 43. Use of the SPIE Macro Instruction

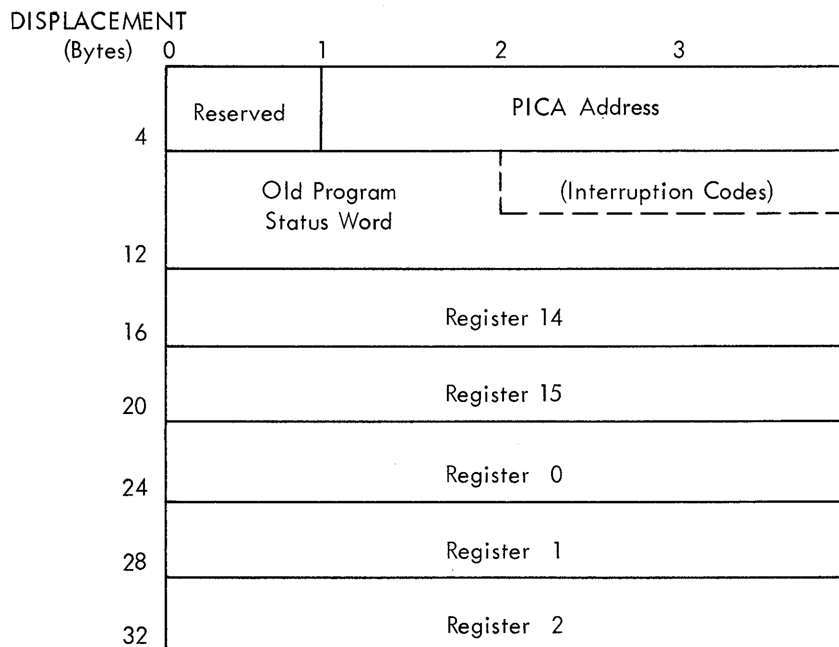


Figure 44. Program interruption element

REGISTER CONTENTS

5.109 When control is passed to the designated exit routine the register contents are as follows:

- Register 0: internal control program information.
- Register 1: address of the program interruption element for the task that caused the interruption.
- Registers 2-12: same as when the program interruption occurred.
- Register 13: address of the save area for the main program (same as when the program interruption occurred). The exit routine must not use this save area.
- Register 14: return address (to the control program).
- Register 15: address of the exit routine.

5.110 The exit routine must be in main storage when it is required, and must return control to the control program using the address passed in register 14. The control program restores registers 14, 15, 0, 1, and 2 from the program interruption element after control is returned, but does not restore the contents of registers 3-13. If a program interruption occurs when the program interruption exit routine is in control, the control program exit routine is given control.

5.111 To determine which type of interruption occurred, the exit routine can interrogate bits 28 through 31 of the old program status word (OPSW) in the program interruption element. The routine can then take corrective action or can simply ignore the exceptional condition.

5.112 The exit routine can alter the contents of the registers when control is returned to the interrupted program. For registers 3 through 13, the routine alters the contents of the actual registers. For registers 14 through 2, the routine alters the contents of the register save area in the program interruption element. This is because the control program

reloads these registers from this area when it returns control to the interrupted program.

5.113 The exit routine can also alter the last four bytes of the OPSW in the program interruption element. By changing the OPSW, the routine can select any return point in the interrupted program.

5.114 The control program returns control to the interrupted program by loading a PSW constructed from the possibly modified OPSW saved in the program interruption element.

SPECIFYING AN ATTENTION EXIT ROUTINE

5.115 If your system (MVT including Model 65 Multiprocessing only) has the time sharing option (TSO), you can use the STAX macro instruction to specify the address of an attention exit routine to gain control when the terminal user strikes the attention key or when the terminal user specifies simulated attention. The details about what you should do in an attention exit routine and how you can use it appear in the Time Sharing Option Guide to Writing a Terminal Monitoring Program or a Command Processor.

PRECISE AND IMPRECISE INTERRUPTIONS

5.116 After an interruption, the old program status word contains the address of the next instruction to be executed in bits 40-63, and the length of the previous instruction in bits 32 and 33. In System/360 Models 65, 67, 75, 85, and 91, and System/370 Models 165 and 195, however, the address of the next instruction may not be precise; if the address is not precise, the instruction length code (ILC) in bits 32-33 is set to zero. You should therefore test the instruction length code for zero before using the next instruction address.

5.117 In Models 65-85, imprecise interruptions can result only from protection and addressing exceptions. In the Model 91, imprecise interruptions result from these and eight other types of exceptions. In the Model 195, imprecise interruptions result from nine other types of exceptions. Figure 45 summarizes the types of program exceptions that can result in an imprecise interruption.

5.118 Except for the protection exception in the Model 91, any exception that can result in an imprecise interruption can also result in a precise interruption. You therefore should not assume that a specific type of exception will always produce an imprecise interruption. Figure 45 defines the conditions under which interruptions are precise in Models 65-195. Note that interruptions are always precise in systems with lower model numbers.

INTERRUPTIONS IN THE MODELS 91 AND 195

5.119 As shown in Figure 46, the interruption code in the Models 91 and 195 differs for precise and imprecise interruptions. For precise interruptions (as for all interruptions in other models), exceptions are indicated in bits 28-31 of the old program status word. For imprecise interruptions, bits 28-31 are zero, and exceptions are indicated in bits 16-27.

5.120 Before testing the interruption code to determine the cause of an interruption, you should test the instruction length code to determine whether the interruption is precise or imprecise. If the instruction length code is zero, indicating an imprecise interruption, you should test bits 28-31 of the old program status word to determine whether the interruption has occurred on a Model 91 or 195. If bits 28-31 are zero, the interruption has occurred on a Model 91 or 195 and the cause of the

Type of Exception	Type of Interruption							
	Precise (ILC≠0)		Imprecise (ILC=0)					
	All Models		Models 65-85 and System/370 Model 165		Model 91		Model 195	
	Bits 16-27	28-31	Bits 16-27	28-31	Bits 16-27	28-31	Bits 16-27	28-31
Operation	(zero)	0001						
Privileged Operation	(zero)	0010						
Execute	(zero)	0011						
Protection	(zero)	0100	(zero)	0100	100000000000	(zero)	100000000000	(zero)
Addressing	(zero)	0101	(zero)[1]	0101[1]	010000000000	(zero)	010000000000	(zero)
Specification	(zero)	0110			001000000000	(zero)		
Data	(zero)	0111			000100000000	(zero)	000100000000	(zero)
Fixed-Point Overflow	(zero)	1000			000010000000	(zero)	000010000000	(zero)
Fixed-Point Divide	(zero)	1001			000001000000	(zero)	000001000000	(zero)
Decimal Overflow	(zero)	1010					000000000010	(zero)
Decimal Divide	(zero)	1011					000000000001	(zero)
Exponent Overflow	(zero)	1100			000000100000	(zero)	000000100000	(zero)
Exponent Underflow	(zero)	1101			000000010000	(zero)	000000010000	(zero)
Siginificance	(zero)	1110			000000001000	(zero)	000000001000	(zero)
Floating-point Divide	(zero)	1111			000000000100	(zero)	000000000100	(zero)
			[1]Except Model 65					

Figure 45. Interruption code in the old program status word

interruption is indicated in bits 16-27. If bits 28-31 are not zero, the interruption has not occurred on a Model 91 or 195, and these bits themselves indicate the cause of the interruption.

5.121 In the Model 91, there are ten types of program exceptions that can cause an imprecise interruption; in the Model 195, there are eleven types. Each is represented by a separate bit in the interruption code (bits 16-27). After an imprecise interruption, the interruption code may indicate more than one type of exception. When it does, the indicated exceptions may be due to a single instruction, or to several instructions whose execution was overlapped. Note that each of the indicated exceptions may have occurred more than once, and there is no indication as to which occurred first.

5.122 If you provide an exit routine to handle any of the exceptions that may result in an imprecise interruption, you should specify all ten such exceptions in the SPIE macro instruction. When an imprecise interruption occurs, your exit routine will be entered only if the PICA indicates all of the exceptions that are indicated in the old program status word. For example, if you provide a routine to handle fixed-point overflow, and if you specify only fixed-point overflow in the SPIE macro instruction, the routine will not be entered if both fixed-point overflow and specification exceptions are indicated for the same interruption.

DECIMAL SIMULATION IN THE MODEL 91

5.123 The instruction set for the model 91 does not include the decimal instructions AP, CP, DP, MP, SP, and ZAP; each of these instructions

Type of Exception	Models 65-85 and System/370 Model 165		Model 91				Model 195		
	Always Precise	Sometimes Precise[1]	Always Precise	Sometimes Precise[2]	Precise in INHIBIT OVERLAP Mode[3]	Precise for Decimal Simulation[4]	Always Precise	Sometimes Precise[5]	Precise in INHIBIT OVERLAP Mode[3]
Operation	X		X				X		
Privileged Operation	X		X				X		
Execute	X		X				X		
Protection		X				X			
Addressing		X		X		X		X	
Specification	X			X		X	X		
Data	X				X	X			X
Fixed-point Overflow	X				X				X
Fixed-point Divide	X				X				X
Decimal Overflow	X					X			X
Decimal Divide	X					X			X
Exponent Overflow	X				X				X
Exponent Underflow	X				X				X
Significance	X				X				X
Floating-point Divide	X				X				X

[1]A protection or addressing exception results in a precise or imprecise interruption, depending on the cause of the exception.
[2]An addressing or specification exception results in a precise or imprecise interruption, depending on the cause of the exception. For details, refer to the Model 91 Functional Characteristics publication.
[3]The indicated interruptions are precise if the INHIBIT OVERLAP switch is set on the system control panel.
[4]The interruption for a protection exception is precise only when simulated by the control program decimal simulator routine. Interruptions for decimal overflow and decimal divide exceptions occur only as simulated interruptions; they do not occur if the control program does not include the decimal simulator routine.
[5]An addressing exception results in a precise or imprecise interruption, depending on the cause of the exception. For details, refer to the Model 195 Functional Characteristics publication.

Figure 46. Precise interruptions in IBM System/360 Models 65, 67, 75, 85, and 91, and System/370 Models 165 and 195

causes an operation exception, which results in a precise interruption. If the decimal simulator routine was specified at system generation, the control program simulates the decimal operation. Otherwise, control is passed to your program interruption exit routine, or to the control program exit routine.

5.124 Decimal simulation may result in an exceptional condition. When it does, the control program simulates a precise interruption as indicated in Figure 46. For decimal overflow, execution is completed and the condition code is set. For other exceptions, execution is suppressed; the condition code and the contents of main storage remain unchanged. Note that the control program does not simulate an interruption for decimal overflow if the interruption is disabled.

EXTENDED-PRECISION FLOATING-POINT SIMULATION

5.125 The OS/360 Extended-Precision Floating-Point Simulator provides full extended-precision arithmetic for all OS users. A divide macro instruction (DXR) is provided for the models that have the extended-precision floating arithmetic facility and all eight instructions are provided for the models that do not. Thus, you can use extended-precision floating-point instructions whether or not your particular machine model has the extended-precision floating-point facility. To do so, write a program-interruption-handling exit routine. The exit routine is required:

- If your machine model already has the extended-precision floating-point facility, and you also wish to use the extended-precision floating-point divide (DXR) macro instruction.
- If your machine model does not have the extended-precision floating-point instructions, but you wish to use these instructions and the extended-precision floating-point divide instruction.

5.126 To determine if the extended-precision floating-point feature is installed in your CPU, call the module IEAXPSIM, which returns a pointer to the appropriate simulator (see "Calling the Simulator," below).

5.127 The format of the extended-precision floating-point divide (DXR) instruction is described in the macro instructions section, and the formats of the other extended-precision floating-point instructions are described in _Principles of Operation_.

EXTENDED PRECISION DIVISION

5.128 To perform extended precision division, use the DXR macro instruction:

```
DXR reg1,reg2
```

where reg1 contains the dividend; reg2 the divisor.

5.129 The first operand (the dividend) is divided by the second operand (the divisor) and is replaced by the normalized quotient. No remainder is preserved. For a discussion of normalization, refer to the section _Floating Point Arithmetic_ in _Principles of Operation_.

Division Process

5.130 The quotient fraction has 28 hexadecimal digits and is developed such that it is the largest number for which the absolute value of the product of the quotient and the divisor fractions is either equal to or less than the absolute value of the adjusted (normalized) dividend fraction. All digits of the dividend and divisor fractions are involved in the operation; the dividend fraction is extended with low-order zeros.

5.131 The sign of the quotient is determined by the rules of algebra; however, if the quotient is made a true zero, its sign is made plus.

5.132 Unless the quotient is made a true zero, the characteristic, sign, and high-order 14 hexadecimal digits of the normalized quotient fraction replace the high-order part of the first operand. The low-order 14 hexadecimal digits of the quotient fraction replace the high-order part of the first operand. The low-order 14 hexadecimal digits of the quotient fraction replace the low-order fraction of the first operand. The low-order sign is made equal to the high-order sign, and the low-order characteristic is made 14 less than the high-order characteristic. However, when the subtraction of 14 causes the low-order characteristic to become less than zero, it is made 128 greater than its correct value. Extended precision arithmetic is further discussed in _Principles of Operation_.

Arithmetic Exceptions

5.133 The following exceptions can occur when using the DXR macro instruction.

- _Exponent overflow_.

- Exponent underflow.

- Floating-point divide.

5.134 Exponent overflow is recognized when the characteristic of the normalized quotient exceeds 127 and the fraction of the quotient is not zero. The operation is completed by making the high-order characteristic 128 less than the current value. If the low-order characteristic also exceeds 127, it is decreased by 128. The quotient fraction and sign remain unchanged. A program interruption for exponent overflow then occurs.

5.135 Exponent underflow is recognized when the characteristic of the normalized quotient is less than zero and neither operand fraction is zero. If the exponent underflow mask bit is set, the operation is completed by making the characteristics of both parts 128 greater than their correct values. The quotient fraction and sign remain unchanged. A program interruption for exponent underflow then occurs. If the exponent underflow mask is zero, a program interruption does not occur; instead, the operation is completed by making both the high-order and low-order parts of the quotient a true zero.

5.136 Exponent underflow is not recognized when the low-order characteristic is less than zero and the high-order characteristic is greater than or equal to zero. Similarly, exponent underflow is not recognized when one or both of the operands underflow during prenormalization, but the quotient can be expressed without encountering underflow.

5.137 The floating-point divide exception is recognized when the divisor fraction is zero. The operation is suppressed, and a program interruption for floating-point divide occurs.

5.138 When the dividend fraction is zero, the quotient is made a true zero, and a possible exponent overflow or underflow is not recognized. A division of zero by zero, however, causes the operation to be suppressed and an interruption for floating-point divide to occur.

5.139 The condition code remains unchanged for all arithmetic exceptions. Figure 47 describes the program interruptions that can occur.

Interruption Type	Description	Action Taken
Operation	The instruction is not installed.	The operation is suppressed.
Specification	Registers other than 0 or 4 are specified, or positions 16-23 do not contain zeros.	The operation is suppressed
Exponent Overflow	The characteristic of the normalized quotient exceeds 127, and neither operand fraction is zero.	The operation is completed.
Exponent Underflow	The characteristic of the normalized quotient is less than zero, neither operand fraction is zero, and the exponent underflow mask bit is set.	The operation is completed.
Floating-Point Divide	The divisor fraction is zero.	The operation is suppressed.

Figure 47. Summary of program interruptions

CALLING THE SIMULATOR

5.140 To use the extended-precision floating-point instructions that your machine model does not have, call the extended-precision floating-point simulator from a program-interruption-handling exit routine. The simulator is a program that is automatically included in your operating system at system generation time. Writing an exit routine to handle program interruptions is discussed under "Program Interruption Processing."

5.141 To use the extended-precision floating-point simulator, specify in the SPIE macro instruction that your exit routine is to receive control if an operation exception occurs. In addition, either your program, during initialization, or the exit routine must perform the following tasks, in this order:

- Prepare a parameter list to pass to IEAXPSIM.
- Pass control to IEAXPSIM, using standard operating system conventions.
- Prepare a parameter list to pass to the simulator.
- Pass control to the simulator, using standard operating system conventions.
- Check the code returned by the simulator.
- Perform corrective action if necessary.

5.142 In addition, your program or the exit routine may perform the following tasks:

- Load the IEAXPSIM module, using the LOAD macro instruction, before its use.
- Delete the IEAXPSIM module, using the DELETE macro instruction, after its use.
- Load the simulator, using the LOAD macro instruction, the first time it is needed.
- Delete the simulator, using the DELETE macro instruction, at the end of the job step.

DESIGNING THE EXIT ROUTINE

5.143 The following paragraphs and Figure 48 should help you design your exit routine. (Figure 48 assumes that the initialization procedures mentioned above are done in the exit routine.)

5.144 The parameter list that you pass to IEAXPSIM must be pointed to by register 1 and must contain a pointer to a doubleword area into which IEAXPSIM will move the name of the simulator module to which you will pass control.

5.145 The parameter list that you pass to the simulator must be pointed to by register 1 and must contain the following:

1. A pointer to the PIE.
2. A pointer to the area containing the contents of general registers 0 through 15 at interrupt time.

3. A pointer to a work area.

4. A pointer to a byte that is nonzero if the last bit of the quotient for a DXR need not be correct.

```
         USING   EXTPRE,15
EXTPRE   STM     3,13,SIMSV+12       Save registers not in PIE
         LR      4,15
         USING   EXTPRE,4            Establish addressability
         DROP    15
         MVC     SIMSV(12),20(1)     Registers 0-2 from PIE
         MVC     SIMSV+56(8),12(1)   Registers 14-15 from PIE
         ST      14,RET              Save return address
         ST      1,PARMB             Pointer to PIE
         LA      13,SAVESIM          Load save area address
         L       15,SIMADD
         LTR     15,15               Does SIMADD contain address?
         BNZ     TOSIM               If so, go directly to simulator
         LINK    EP=IEAXPSIM,PARAM=(PARMA)
         LOAD    EPLOC=SIMUL         Load simulator
         LR      15,0                Put simulator's address in
                                       register
         ST      0,SIMADD            Save address of simulator
TOSIM    LA      1,PARMB             Parameter list address
         BALR    14,15               Go to simulator
         LTR     15,15               Error or exceptional condition?
         ...

*HERE THE EXIT ROUTINE SHOULD DETERMINE THE ERROR OR THE
*EXCEPTIONAL CONDITION THAT OCCURRED IN SIMULATING AND
*TAKE APPROPRIATE ACTION.

         ...
         B       OUT
GOODOUT  EQU     *

*HERE THE EXIT ROUTINE SHOULD TAKE APPROPRIATE ACTION WHEN
*NO ERROR OR EXCEPTIONAL CONDITION OCCURRED DURING SIMULATION.

         ...
OUT      L       14,RET
         LM      3,13,SIMSV+12       Restore registers
         BR      14                  Return

*WHEN THE EXIT ROUTINE NO LONGER NEEDS THE SIMULATOR,
*THE ROUTINE SHOULD DELETE IT.

         ...
         DELETE  EPLOC=SIMUL
         ...
PARMA    DC      X'80',AL3(SIMUL)    Pointer to simulator name
SIMUL    DS      D                   Simulator name
PARMB    DS      F                   For pointer to PIE
         DC      A(SIMSV)            Address of register area
         DC      A(WORK)             Address of work area
         DC      X'80',AL3(ZERO)     Divide adjust switch pointer
ZERO     DC      X'0'                Adjust switch for divide
WORK     DC      50D                 Work area
SIMSV    DS      16F                 Register area
SIMADD   DC      F'0'                Address of simulator
RET      DS      F                   Return address
SAVESIM  DS      18F                 Save area
```

Figure 48. Calling the extended-precision floating-point simulator

5.146 The work area must be at least 30 doublewords (240 bytes) if your installation's machine model has the extended-precision floating-point facility or at least 50 doublewords (400 bytes) if it does not. The exit routine shown in Figure 48 can be used for either type machine model because its work area is 50 doublewords.

5.147 To obtain the name of the extended-precision floating-point simulator installed in your system, call the module IEAXPSIM, which returns a pointer to the name of the simulator in the doubleword that you provide. In Figure 48, the doubleword is SIMUL.

5.148 Before passing control to the simulator, you can use the LOAD macro instruction to bring the simulator into main storage if it is not already there. The entry point name is specified as the name returned from IEAXPSIM. After issuing LOAD, you can pass control to the simulator, using standard calling conventions.

5.149 Upon regaining control from the simulator, the exit routine should check register 15 for one of the two return codes shown in Figure 49.

5.150 If the return code was X'FF', the exit routine determines the kind of error encountered by the simulator by examining the interruption code in bits 28-31 of the PSW. Figure 50 shows the possible settings of the interruption code.

5.151 The simulator will adjust the condition code in the old PSW in the PIE (bits 34-35) to indicate the result of an AXR or SXR macro instruction. When a program interruption occurs within the simulator while fetching the argument of the MXD macro instruction, the instruction address in the PSW in the PIE is restored to its setting at operation-interrupt time.

5.152 The simulator never alters the Program Check Old PSW at location 40. Its interruption code will be an operation exception except for the MXD macro instruction, when it may be a protection, addressing, or specification exception.

5.153 The simulator should be deleted by the using program if it was obtained via the LOAD macro instruction.

5.154 If the full simulator (IEAXPALL) is loaded on a CPU that already has the extended-precision floating-point facility, no abnormal conditions will result. Only the DXR macro instruction will be simulated. However, the simulation of the DXR function is slower than if the IEAXPDXR were used, since the other extended-precision operations in the divide algorithm are also simulated.

5.155 If IEAXPDXR is loaded on a CPU without the extended-precision floating-point facility, a 0C1 ABEND will occur when an extended-precision divide is simulated. In the simulation of the other extended-precision macro instructions, a return code of X'FF' is passed to the caller and no simulation is attempted.

Hexadecimal Code	Meaning
00	The operation was successful.
FF	The operation was not successful, or an exceptional condition occurred.

Figure 49. Return codes from the extended-precision floating-point simulator

Meaning of Interruption	Bits 28-31
The simulator found that the operation was not an extended-precision floating-point operation and returned control without further processing.	0001
Protection exception [1] [3]	0100
Addressing exception [1] [3]	0101
Specification exception [1] [2] [3]	0110
Exponent overflow exception [4]	1100
Exponent underflow exception [4]	1101
Significance exception [4]	1110
Floating-point divide [4]	1111

[1]When the simulator encounters these exceptions, it stops processing and returns control to the exit routine.
[2]An incorrect extended-precision floating-point register was specified, the third byte of the DXR macro instruction was not X'00' or a register other than 0 or 4 was specified in the R1 or R2 field of the DXR macro instruction.
[3]The error occurred during the processing of an MXD macro instruction.
[4]The error occurred during simulation.

Figure 50. Interruption codes returned by the simulator

ABNORMAL CONDITION HANDLING

5.156 It is not possible to provide procedures for all possible conditions which can occur during the execution of a program. You should, of course, be sure that you can process all valid data, and that your program satisfies all the requirements of the problem. The more general you make the program, the greater the number of additional routines you will require to handle special cases. But you will not be able to provide routines to detect and correct all of the special or abnormal conditions that can occur.

5.157 The control program does a great deal of checking for abnormal conditions. A standard program interruption routine is provided to detect and process errors such as protection violations or addressing errors. The data management and supervisor routines provide some error checking facilities to ensure that, based on the information you have provided, only valid data is being processed, and that no requests with conflicting requirements have been made. For the abnormal conditions that can possibly be corrected, control is returned to your program with a return code indicating the probable source of the error. For conditions that indicate that further processing would result in degradation of the system or destruction of existing data, the control program abnormal termination routine is given control.

5.158 There will be abnormal conditions unique to your program, of course, that the control program cannot detect. Figure 51 is an example of one of these. The routine shown in Figure 51 checks a control field in an input parameter list to determine which function the program is to perform. Only characters between 1 and 4 are valid in the control field. The presence of any other character is invalid, but the routine must be

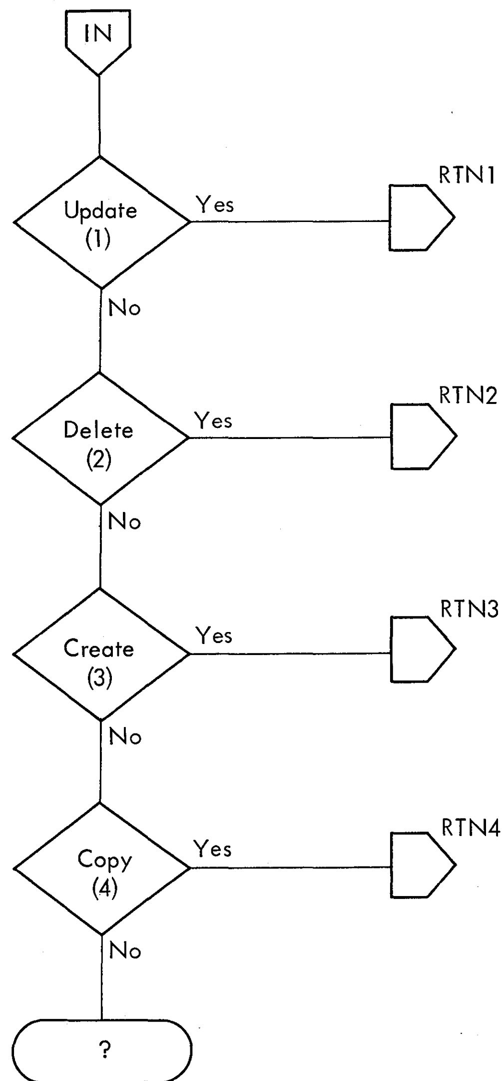


Figure 51. Abnormal condition detection

prepared to detect and handle these characters. The routine should indicate its inability to continue processing by returning control to the calling program with an error return code. The calling program should then try to interpret the return code and to recover from the error. If it cannot do so, the calling program should detach its incomplete subtasks, execute its usual termination procedures, and return control to its calling program, again with an error return code. This procedure may result in termination of all the tasks of a job step; if it does, the COND parameters of the JOB and EXEC statements may be used to determine whether or not subsequent job steps should be executed.

5.159 An alternative to this procedure is to pass control to the control program abnormal termination routine by issuing an ABEND macro instruction. This alternative is simpler, but it offers less opportunity for error recovery and continued processing unless a STAE macro instruction, specifying a STAE exit routine address, is issued to override the ABEND. The abnormal termination facilities available through the use of the ABEND macro instruction are discussed below; an explanation of the facility to intercept abnormal termination through the STAE macro instruction is presented following the ABEND discussion.

5.160 The position within the job step hierarchy of the task for which the ABEND macro instruction is issued determines the exact function of the abnormal termination routine.

5.161 If an ABEND macro instruction is issued when the job step task (the highest level or only task) is active, or if the STEP operand is coded in an ABEND macro instruction issued during the performance of any task in the job step, all the tasks in the job step are terminated. An ABEND macro instruction (without a STEP operand) that is issued in performance of any task other than the job step task usually causes only that task and the subtasks of that task to be abnormally terminated. However, if the abnormal termination cannot be fulfilled as requested, it may be necessary for the supervisor to abnormally terminate the job step task. The most frequent cause of this is that the subtask does not have sufficient main storage for ABEND's processing. ABEND "steals" main storage allocated to the job step task and needed by it to continue normal processing. The abnormal termination routine works in the same manner whether it is given control from the control program or a problem program.

5.162 When a task is abnormally terminated, the control program performs the following functions:

- Lowers the responsibility counts for the load modules brought into main storage during the performance of the task.
- Releases the main storage subpools owned by the task.
- Cancels the time interval if one had been established for the task.
- Issues a CLOSE macro instruction for any data control blocks which were opened during the performance of the task.
- Purges any outstanding input or output requests.
- Cancels any requests for operator replies made using a WTOR macro instruction.
- Cancels any requests for resources made using an ENQ macro instruction.

5.163 If the job step <u>is not</u> to be terminated, the following action is taken:

- The abnormal termination functions listed above are performed, starting with the lowest level task, for each of the subtasks of the task which was active when the ABEND macro instruction was issued. A DETACH macro instruction is issued by the control program for each of the subtasks.
- The completion code specified in the ABEND macro instruction is placed in the task control block of the active task (the task for which the ABEND macro instruction was issued).
- If the ECB operand was designated in the ATTACH macro instruction issued to create the active task, the completion code specified in the ABEND macro instruction is placed in the designated event control block, and the completion bit is turned on.
- If the ETXR operand was designated in the ATTACH macro instruction issued to create the active task, the end-of-task exit routine is scheduled to be given control when the originating task becomes active.
- If neither the ECB nor ETXR operands were designated when the ATTACH macro instruction was issued, a DETACH macro instruction is issued by the control program for the active task.

5.164 If the job step is to be terminated, the following action is taken:

- The abnormal termination functions listed above are performed, starting with the lowest level task, for all tasks in the job step. All main storage belonging to the job step is released. None of the end-of-task exit routines are given control.

- The completion code specified in the ABEND macro instruction is written on the system output device.

- Unless you specify otherwise in your job control statements, the remaining job steps in the job are skipped. However, the statements defining these steps are checked for proper syntax.

5.165 In any operating system, it is possible to restart a job step that has been abnormally terminated. Restart can occur either at the beginning of the job step or at an internal checkpoint. A detailed discussion of checkpoint and restart appears in the publication Advanced Checkpoint/Restart.

INTERCEPTING ABNORMAL TERMINATION OF TASKS

5.166 Abnormal termination of a task can be intercepted through the use of the STAE macro instruction. When a task that has previously issued a STAE macro instruction is scheduled for abnormal termination, termination processing is intercepted and control is returned to the user at his STAE exit routine address, as specified in the STAE macro instruction. Within the STAE exit routine, the user can perform pre-termination functions or diagnose the error. He can also determine whether abnormal termination should continue for the task, or whether a STAE retry routine, which would circumvent abnormal termination, should be scheduled. For further information on scheduling a STAE retry routine, see the MFT or MVT Guide.

5.167 At the time the abnormal termination is scheduled, the STAE exit routine must be resident. It must either be part of the program issuing STAE or be brought into storage via the LOAD macro instruction.

5.168 The STAE exit routine can contain an ABEND macro instruction, but it must not contain a STAE or an ATTACH macro instruction.

5.169 A single user program can issue more than one STAE macro instruction with the create (CT) operand. Each issuance makes the previous STAE environment temporarily inactive. The suspended STAE environment can be reestablished by canceling the current STAE. Unless it is intended that the existing STAE environment be saved, it should be canceled prior to issuing another STAE. Otherwise, main storage will be wasted by STAE control blocks for inactive STAE environments.

5.170 If the user wishes to use the same exit routine for several tasks at the same time, it must be reenterable. For convenience sake, it is recommended that all STAE exit routines be reenterable.

5.171 The user can cancel (make the previous STAE request active) or overlay the current STAE request. The STAE request that is canceled or overlaid is the one most recently made. If no STAE requests are active for the task at the time a cancel or overlay is issued, or if the user attempts to cancel or overlay a STAE request not associated with his Request Block level of control, he will be informed that his request is invalid by a return code. A STAE request can be canceled by issuing the STAE macro instruction with the STAE exit routine address specified as zero. Overlaying is done by issuing a STAE macro instruction specifying OV.

```
         ...
         STAE  EXIT1,CT,PARAM=LIST1,                              C
               XCTL=YES,ASYNCH=YES,                               C
               PURGE=QUIESCE        Initial STAE request
         ...
         ...
         LA    5,EXIT2              Put new exit routine address in
                                      register 5
         STAE  (5),OV,PURGE=NONE    STAE request for overlay
         ...
LIST1    DC    F'0'                 Parameter list for exit routines
         DC    X'A0'
EXIT1    EQU   *                    Entry point of first exit routine
               .
               .
               .
EXIT2    EQU   *                    Entry point of second exit routine
               .
               .
               .
```

Figure 52. Use of STAE macro instruction

5.172 When a program issuing STAE returns control to a previous level via an SVC 3, all STAE requests issued by that program are canceled. A STAE request specifying XCTL=YES is not canceled when the STAE user issues an XCTL macro instruction and the STAE environment is connected to the program in control after XCTL. If a program terminates by any means other than an SVC 3 or a RETURN macro instruction, all STAE requests must be canceled by the terminating program before returning control to another program.

5.173 STAE requests issued by a program are queued for that program so that the last STAE request issued is the active one, that is, it is the one that causes the STAE exit routine to receive control if the program is abnormally terminated. If the active STAE request is canceled, the next-to-the-last STAE request becomes the last and thus the active one.

5.174 Figure 52 shows the use of the STAE macro instruction. The STAE request is initially made specifying a STAE exit routine address (EXIT1) and parameter list address (LIST1). The XCTL=YES parameter indicates that this STAE request will not be canceled if the program terminates via the XCTL macro instruction. The ASYNCH=YES parameter indicates that asynchronous interruptions will be allowed during STAE exit routine processing. The PURGE=QUIESCE parameter indicates that input/output requests not yet performed are removed from the system's active input/output queue (purged) but can later be returned to that queue (restored). If PURGE=QUIESCE cannot be honored by the system, the input/output requests are removed from the queue with the halt option and therefore cannot be restored.

5.175 In the second issuance of STAE, the previous STAE request is modified through the overlay (OV) option. The STAE exit routine address is now EXIT2, and input/output intervention will now be bypassed, but the parameter list address remains the same. Because the XCTL and ASYNCH operands were not specified in the STAE macro instruction specifying OV, the default values XCTL=NO and ASYNCH=NO are assigned, replacing the XCTL=YES and the ASYNCH=YES specified in the previous STAE. Note that the overlaid STAE macro instruction cannot be reestablished by issuing a cancel STAE.

5.176 After a STAE macro instruction has been issued, the register contents upon return to the user are as follows:

- Registers 0, 1: Unpredictable.
- Registers 2-13: Same as when STAE was issued.
- Register 14: Unpredictable.
- Register 15: Completion code.

Hexadecimal Code	Indication
00	Successful completion of creating, overlaying, or canceling a STAE request.
04	No storage obtainable for a STAE request.
08	A STAE request to be canceled or overlaid did not exist, or a STAE was issued in the user's exit routine.
0C	Invalid exit routine or parameter list address.
10	Attempt to cancel or overlay another user's STAE request.

5.177 When a program with an active STAE request encounters an ABEND situation, control is passed to the STAE exit routine. ABEND processing continues and the STAE exit routine does not receive control in the following situations:

- If the abnormal termination is caused by an operator's CANCEL, job step timer expiration, or the detaching of an incomplete task.
- If the terminating task is in must complete status and problem program mode. (Putting a task in the must complete status is explained in the _MFT_ and _MVT Guides_.)
- If the OUTLIMIT is exceeded for SYSOUT.
- If an invalid ABEND recursion (an abnormal condition encountered during abnormal termination) occurs.
- If an abnormal condition is encountered during normal termination.
- If the failing task has been in a wait state for more than 30 minutes.
- If the STAE macro instruction was issued by a subtask and the mother task abnormally terminates.
- If the exit routine was specified for a subtask, via the STAI operand of the ATTACH macro instruction, and the mother task abnormally terminates.
- If the abnormal termination is because the task that issued the STAE still has active subtasks when it returns to the control program via an SVC 3.
- If any other problem arises while the control program is preparing to give control to the STAE exit routine.

5.178 Before the STAE exit routine receives control, any existing SPIE requests are canceled and the purge request specified in the STAE macro

instruction is fulfilled. The register contents upon entry to the STAE exit routine are as follows:

- Register 0:

Hexadecimal Code	Indication
00	Active I/O at the time of the ABEND was quiesced and is restorable.
04	Active I/O at the time of the ABEND was halted and is not restorable.
08	No I/O was active at the time of the ABEND.
0C	No work area was obtained.
10	No I/O processing was requested.

- Register 1: Address of a 104-byte work area, as shown in Figure 53.
- Registers 2-12: Unpredictable.
- Register 13: Address of a supervisor-provided register save area.
- Register 14: Return address.
- Register 15: Address of the STAE exit routine.

Offset			
0	Address of STAE exit routine parameter list or 0	Flags	System and user completion codes
8	PSW at time of ABEND		
16	Last problem program PSW before ABEND		
24	Contents of registers 0-15 at time of ABEND (64 bytes)		

If a problem program issued STAE:

Offset		
88	Name of abnormally terminated program or 0	
96	Address of entry point to abnormally terminated program if ABEND occurred in program represented by PRB; otherwise zero	0

If supervisor program issued STAE:

Offset		
88	Address of request block of abnormally terminated program	0
96	0	

Figure 53. Work area for STAE exit routine

5.179 Note: Registers 13 and 14, if used by the STAE exit routine, must be saved and restored prior to returning to the calling program. Standard subroutine linkage conventions apply.

Bytes 4-7 in Figure 53 are used as follows:

Bit	Content	Indication
0	1	Dump to be given.
0	0	Dump not to be given.
1	1	Job step to be terminated.
1	0	Only failing task to be terminated.
2-7	--	Not used.
8-19	--	System completion code (packed, unsigned, hexadecimal).
20-31	--	User completion code (hexadecimal).

5.180 If main storage was not available for the work area, the register contents upon entry to the STAE exit routine are as follows:

- Register 0: 12 (decimal)
- Register 1: Flags and completion codes (see Figure 53, bytes 4-7 for format).
- Register 2: Address of STAE exit parameter list.
- Register 3-13: Unpredictable.
- Register 14: Return address.
- Register 15: Exit routine address.

5.181 Note: If a work area could not be provided by the control program, a register save area will not be provided either. A save area is never provided for a retry routine.

5.182 Before returning control to the operating system from the STAE exit routine, the user must put a return code in register 15. The return code indicates whether ABEND processing is to be continued for the task or whether a STAE retry routine should be scheduled. (The details about scheduling a STAE retry routine are in the MFT and MVT Guides.)

5.183 The return codes to be placed in register 15 are defined as follows:

Hexadecimal Code	Indication
00	No retry routine provided.
04	A STAE retry routine has been provided and the Request Block chain should be purged.
08	A STAE retry routine has been provided and the Request Block chain should not be purged. (To be used by routines in supervisor state only.)
0C	A STAI (Subtask ABEND intercept) retry routine has been provided.
10	No further STAI processing; ABEND processing is to continue.

For further information on the option of STAE retry, see the MFT or MVT Guide.

INTERCEPTING ABNORMAL TERMINATION OF SUBTASKS

5.184 To provide an exit in your program to intercept abnormal termination of a subtask, use the STAI (subtask ABEND intercept) operand of the ATTACH macro instruction you issue to create the subtask. The STAI request issued for any subtask will be propagated for all subtasks further down the tree. For example, Task A attaches Task B and uses the STAI operand on the ATTACH macro instruction. When Task B attaches Task C, the STAI request issued by A will be active for C as well as B. When a task abnormally terminates, any STAE exit routine specified for it receives control first. Then any STAI exit routines specified receive control, beginning with the last specified STAI exit routine.

5.185 Since more than one subtask may abnormally terminate at the same time, the STAI exit routine may be used by more than one task concurrently. Therefore, the exit routine must be reenterable, or it may fail during the second entry.

THE DUMP

5.186 There are three types of main-storage dumps produced by the operating system:

- A dump obtained through use of the DUMP operand in the ABEND macro instruction.
- A dump obtained through use of the SNAP macro instruction.
- A core image dump, produced in the event of a failure by a system routine.

5.187 You can request a dump by using the ABEND or SNAP macro instruction. You cannot request the core image dump -- it is produced automatically by the system whenever a failure occurs in a system routine.

ABEND AND SNAP DUMPS

5.188 When the dump is requested using an ABEND macro instruction, no further processing is performed for the active task; use of the SNAP macro instruction allows the task to continue after the completion of the dump. The control program generally requests a dump for you when it issues an ABEND macro instruction.

5.189 The data set containing the dump can reside on any device which is supported by the basic access technique using sequential organization (BSAM). The dump is placed in the data set described by the DD statement you provide. If a printer is selected the dump is printed immediately. However, if a direct access or tape device is designated, a separate job is scheduled to obtain a listing of the dump, and to release the space on the device.

5.190 The format of the dump is shown in the publication Programmer's Guide to Debugging. The entire dump shown in that publication is provided in an abnormal termination dump if a DD statement with a ddname of SYSABEND is provided; only the problem program areas and system control blocks associated with the problem program are dumped if a DD statement with a ddname of SYSUDUMP is provided. Use of the SNAP macro instruction allows you to request only selected portions of the entire dump for any task in the job step; the format of the portions selected is the same as the format of the same portions of an abnormal termination dump.

5.191 When an abnormal termination dump is requested, the entire dump is provided for the active task, along with a dump of the control blocks and save area for each of the higher level tasks which are predecessors of the active task being terminated and for each of the subtasks of the active task. The control program dump routine uses the addresses you stored in words 2 and 3 of each save area to follow the "chain" of save areas provided by each calling program in each task. If an ABEND macro instruction was issued when task B1 (Figure 29) was active, for example, a complete dump would be provided for task B1. The control blocks and save areas for task B, task B1a, and the job step task would also be provided in separate dumps.

To get a dump:

5.192
- You must provide a DD statement for each job step in which a dump is requested. For an abnormal termination dump, the ddname must be SYSABEND or SYSUDUMP; for a SNAP macro instruction dump, the ddname must be any name except SYSABEND or SYSUDUMP. The requirements for

writing the DD statement are described in the Programmer's Guide to Debugging.

5.193 • To obtain a dump using the SNAP macro instruction, you must provide a data control block, and issue an OPEN macro instruction for the data set before any SNAP macro instructions are issued. The data control block must contain the following parameters: DSORG=PS, RECFM=VBA, MACRF=(W), BLKSIZE=nnn, and LRECL=125, where nnn is 882 for MFT and either 882 or 1632 for MVT. (The data control block is discussed in the Data Management Services manual.) If your program is to be processed by the loader, you should also issue a CLOSE macro instruction for the SNAP data control block.

5.194 • Sufficient unused main storage must be available in the area assigned to the job step to hold the control program dump routine and, if not already in main storage, the BSAM data management routines. For an abnormal termination dump, additional main storage is required for the routines to process the OPEN macro instruction issued by the control program, and for the trace table. Refer to the Storage Estimates publication for storage requirements.

INDICATIVE DUMP

5.195 In an operating system with MFT, you can obtain an indicative dump, as shown in the Programmer's Guide to Debugging. This dump is provided in response to a request for an abnormal termination dump when either you did not provide a DD statement with the ddname SYSABEND or SYSUDUMP, or the control program entry for that DD statement was destroyed. The indicative dump is printed on the system output device. The indicative dump is not provided in an operating system with MVT.

CORE IMAGE DUMP

5.196 If a system routine fails, the system automatically supplies a dump of main storage. This dump, called the core image dump, provides diagnostic information. The system writes the core image dump in the system data set SYS1.DUMP or in a tape volume at the device designated when the operating system was initially loaded.

5.197 In systems with MFT or MVT, use the IMDPRDMP Service Aid program to obtain a printout of the dump. A description of IMDPRDMP and the core image dump formats appear in the Service Aids publication.

5.198 For guidance in using the core image dumps from all configurations of the operating system, refer to the Programmer's Guide to Debugging.

6.1 No matter which configuration of the operating system you are using, there is a finite amount of main storage available to your job step. You have a partition (MFT) or region (MVT) of fixed size available to your job step.

6.2 In an operating system with MFT, the main storage available to problem programs is divided into 1 to 15 fixed partitions. The division is made during system generation, but the operator can enlarge a partition by combining it with others. Each partition is associated with one or more "job classes," which can be varied by the operator. On the basis of job class and priority specified in a JOB statement, a job is assigned to a partition and scheduled for execution. A job step will be abnormally terminated if it requires more main storage than is available in the partition.

6.3 In a system with MVT, available main storage is divided into regions, which vary in size and number according to the requirements of the job steps being performed. Job steps are selected for execution according to job class and priority, and each is assigned a region of the size specified in a JOB or EXEC statement. If the highest priority job step requires a larger region than can be made available, its execution is delayed, and a lower priority job step (one with sufficiently lower storage requirements) is initiated. After a job step has been initiated, its region can be extended only if the rollout/rollin option has been included in the system. (For a description of rollout/rollin, refer to the MVT Guide.)

6.4 You obtain the use of the main storage area assigned to your job step through implicit and explicit requests for main storage. The use of a LINK macro instruction is an implicit request for main storage; the control program allocates space before bringing the load module into your job pack area. The use of the GETMAIN macro instruction is an explicit request for a certain number of bytes of main storage to be allocated to the active task. In addition to your requests for main storage, requests are made by the control program and data management routines for areas to contain some of the control blocks required to manage your tasks.

6.5 The following paragraphs discuss some of the techniques that can be applied for efficient use of the main storage area reserved for your job step. These techniques apply as well to the data management portions of your programs. The specific data management main storage allocation facilities are discussed in Data Management Services; the principles discussed here provide the background you will need to use these facilities.

EXPLICIT REQUESTS

6.6 Main storage can be explicitly requested for the use of the active task by issuing a GETMAIN macro instruction. The main storage request is satisfied by allocating a portion of the main storage area reserved for the job step to the active task. You cannot use the main storage area reserved for the job step without first requesting it; if you attempt to use it without requesting it, the task is abnormally terminated. The main storage area is not set to zero when allocated.

6.7 You return control of main storage by issuing a FREEMAIN macro instruction. This does not release the area from control of the job step; it only makes the area available to satisfy the requirements of

additional requests for any task in the job step. The main storage assigned to a task is also released for other uses when the task terminates, except as indicated under "Subpool Handling."

SPECIFYING LENGTHS

6.8 Main storage areas are always allocated to the task in multiples of eight bytes and begin on a doubleword boundary. The request for main storage is given in terms of bytes; if the number specified is not a multiple of eight, it is rounded to the next higher multiple of eight. You can make repeated requests for a small number of bytes as you need the area or you can make one large request to completely satisfy the requirements of the task. There are two reasons for making one large request: it is the only way you can be sure of getting contiguous storage area and, because you only make one request, the amount of control program overhead is less.

TYPES OF EXPLICIT REQUESTS

6.9 There are four methods of explicitly requesting main storage using a GETMAIN macro instruction. Each of the methods, which are designated by coding an associated character in the operand field of the GETMAIN macro instruction, has certain advantages, depending on the requirements of your program. The last three methods do not produce reenterable code unless coded in the list and execute forms as indicated in the paragraph "Implicit Requests." The methods are as follows:

6.10 REGISTER TYPE (R): Specifies a request for a single area of main storage of a specified length. The address of the area is returned in register 1. This type of request produces reenterable code, because parameters are passed to the control program in registers, not in a parameter list.

6.11 ELEMENT TYPE (E): Specifies a request for a single area of main storage of a specified length. The control program places the address of the allocated area in a fullword you supply.

6.12 LIST TYPE (L): Specifies a request for one or more areas of main storage. You place the length of each area in a list; each list entry represents a request for one area of main storage. The control program places the addresses of the allocated areas in consecutive full words in another list you supply. The addresses are placed in the list in the same order they were requested. This type of request can be made only in an operating system with MVT.

6.13 VARIABLE TYPE (V): Specifies a request for a single area of main storage with a length between two values you specify. The control program will attempt to allocate the maximum length you specify; if not enough storage is available to allocate the maximum length, the largest area with a length between the two values is allocated. The control program places the address of the area and the length allocated in two consecutive fullwords you supply.

6.14 In addition to the above methods of requesting main storage, you can designate the request as conditional or unconditional. (A register type request is always unconditional.) If the request is unconditional and sufficient main storage is not available to fill the request, the active task is abnormally terminated. If the request is conditional, however, and insufficient main storage is available, a return code of four is provided in register 15; a return code of zero is provided if the request was satisified. When a conditional list-type request is made, no main storage is allocated unless all of the requested areas can be allocated.

```
         ...
         GETMAIN  EC,LV=16000,A=ANSWADD,  Conditional request for
                  HIARCHY=0                16000 bytes in processor
                                           storage
         LTR      15,15                   Test return code
         BZ       PROCEED1                If 16000 bytes allocated,
                                           proceed
         DELETE   EP=REENTMOD             If not, free main storage
         GETMAIN  VU,LA=SIZES,A=ANSWADD,  Try to get smaller
                  HIARCHY=0                amount in processor
                                          storage
         L        4,ANSWADD+4             Load and test allocated
                                           length
         CH       4,MIN                   If 8000 or more, use
                                           procedure 1
         BNL      PROCEED1                If less than 8000, use
                                           procedure 2
PROCEED2 ...
PROCEED1 ...
MIN      DC       H'8000'                 Min. size for procedure 1
SIZES    DC       F'4000'                 Min. size to proceed at
                                           all
         DC       F'16000'                Size of area for maximum
                                           efficiency
ANSWADD  DC       F'0'                    Address of allocated area
         DC       F'0'                    Size of allocated area
```

Figure 54. Use of the GETMAIN macro instruction

6.15 An example of the use of the GETMAIN macro instruction is shown in Figure 54. The example assumes a program which operates most efficiently with a work area of 16,000 bytes, with a fair degree of efficiency with 8000 bytes or more, inefficiently with 4000 to 8000 bytes, and not at all with less than 4000 bytes. The program uses a reenterable load module with an entry point name of REENTMOD, and will use it again later in the program; to save time, the load module was brought into the job pack area using a LOAD macro instruction so that it would be available when it was required.

6.16 A conditional request for a single element of main storage with a length of 16000 bytes is requested in Figure 54. The return code in register 15 is tested to determine if the area was available; if the return code was zero (the 16,000 bytes were allocated), control is passed to the processing routine. If sufficient area was not available, an attempt to obtain more main storage area is made by issuing a DELETE macro instruction to free the area occupied by the load module REENTMOD. A second GETMAIN macro instruction is issued, this time an unconditional request for an area between 4000 and 16000 bytes in length. If the minimum size is not available, the task is abnormally terminated. If at least 4000 bytes were available, however, the task can continue. The size of the area actually allocated is determined and one of the two procedures (efficient or inefficient) is given control.

SUBPOOL HANDLING (IN MFT SYSTEMS WITHOUT SUBTASKING)

6.17 There is only one unnumbered subpool in an operating system with MFT. In this configuration of the operating system all main storage requests are satisfied by allocating storage from this unnumbered subpool. If subpool numbers are specified, the numbers are ignored if they are not greater than 127 (the greatest number that is valid in a system with MVT). If subpool numbers greater than 127 are specified, the job step is abnormally terminated.

SUBPOOL HANDLING (IN MFT SYSTEMS WITH SUBTASKING)

6.18 Although subpools are not created in MFT systems, it is convenient to call the partition itself "subpool 0." That is, all main storage in a partition is shared by all tasks active in that partition. Main storage not allocated to any task is called "free storage." "Subpool 240" is used by the supervisor to enable the sharing of a reenterable program invoked by a LOAD macro instruction. "Subpool 255" is used by the supervisor to request storage from the system queue area. User programs may request main storage from the partition by specifying any subpool number from 0 to 127 or by specifying no number at all (this provides compatibility with MVT). User-program implied requests for storage, initiated when the user executes an ATTACH, LINK, LOAD, or XCTL macro instruction, are recorded by the supervisor in order for the storage to be freed during termination.

SUBPOOL HANDLING (IN MVT SYSTEMS)

6.19 In an operating system with MVT, subpools of main storage are provided to assist in main storage management and for communications between tasks in the same job step. Because the use of subpools requires some knowledge of how the control program manages main storage, a discussion of main storage control is presented here.

MAIN STORAGE CONTROL

6.20 When the job step is given a region of main storage, all of the storage area available for your use within that region is unassigned. Subpools are created only when a GETMAIN macro instruction is issued designating a subpool number. If no subpool number is designated, the main storage is allocated from subpool 0, which is created for the job step by the control program when the job step task is initiated.

6.21 Note: If main storage is allocated to a subtask by the user program while the system is executing in the supervisor state or with a protection key of 0, no other task should free that main storage. If some other task does free that main storage, you get unpredictable results.

6.22 For purposes of control and main storage protection, the control program considers all main storage within the region in terms of 2048-byte blocks. These blocks are assigned to a subpool, and space within the blocks is allocated to a task, by the control program when requests for main storage are made. When there is sufficient unallocated main storage within any block assigned to the designated subpool to fill a request, the main storage is allocated to the active task from that block. If there is insufficient unallocated main storage within any block assigned to the subpool, a new block (or blocks, depending on the size of the request) is assigned to the subpool, and the storage is allocated to the active task. The blocks assigned to a subpool are not necessarily contiguous unless they are assigned as a result of one request. Only blocks within the region reserved for the associated job step can be assigned to a subpool.

6.23 Figure 55 is a simplified view of a main-storage region containing four 2048-byte blocks of storage. All the requests are for main storage from subpool 0. The first request from some task in the job step is for 504 bytes; the request is satisfied from the block shown as Block A in the figure. The second request, for 2000 bytes, is too large to be

satisfied from the unused portion of Block A, so the control program assigns the next available block, Block B, to subpool 0, and allocates 2000 bytes from Block B to the active task. A third request is then received, this time for 1000 bytes. There is not sufficient unallocated area remaining in Block B (blocks are checked in the order last in, first out), but there is enough space in Block A, so an additional 1000 bytes are allocated to the task from Block A. Because all tasks may share subpool 0, Request 1 and Request 3 do not have to be made from the same task, even though the areas are contiguous and from the same 2048-byte block. Request 4, for 3000 bytes, requires that the control program allocate the area from 2 contiguous blocks which were previously unassigned, Block D and Block C. These blocks are assigned to subpool 0.

6.24 As indicated in the preceding example, it is possible for one 2048-byte block in subpool 0 to contain many small areas allocated to many different tasks in the job step, and it is possible that numerous blocks could be split up in this manner. Areas acquired by a task other than the job step task are not released automatically on task termination.

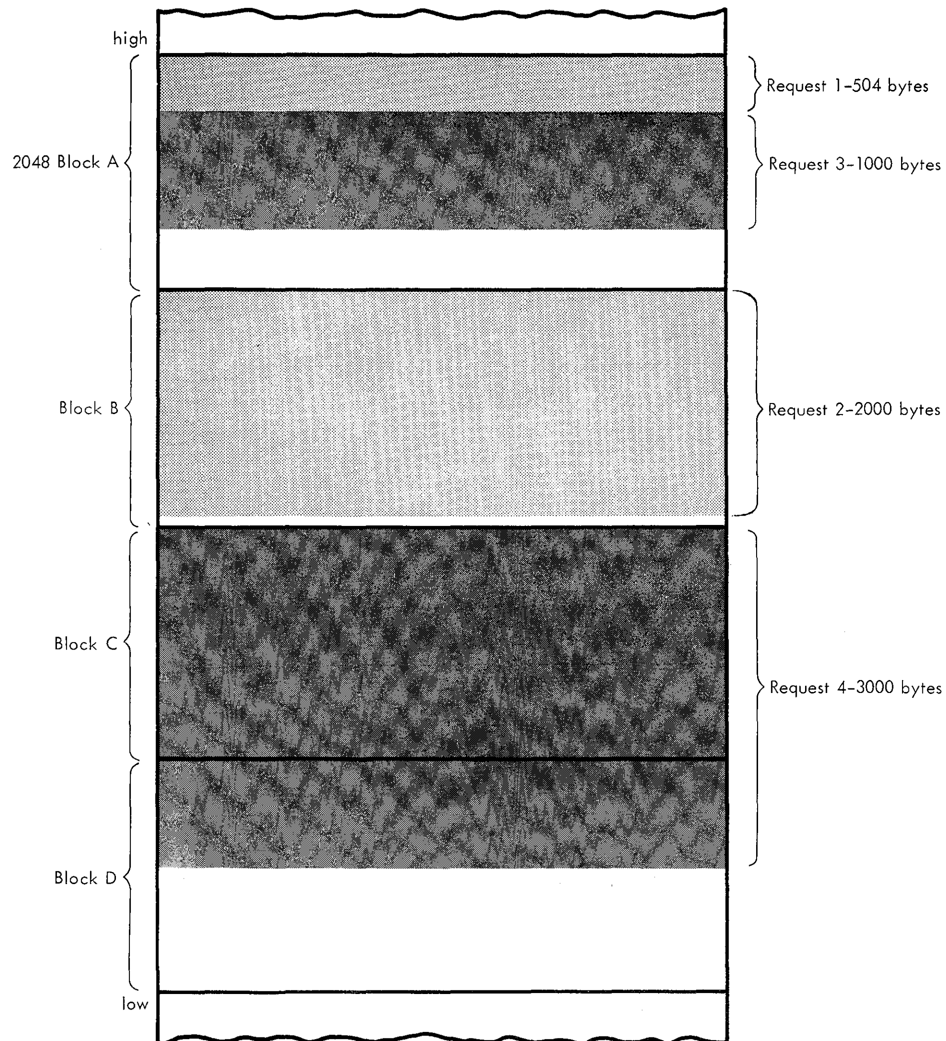


Figure 55. Main-storage control

Even if FREEMAIN macro instructions were issued for each of the small areas before a task terminated, the probable result would be that many small unused areas would exist within each block, while the control program would be continually assigning new blocks to satisfy new requests. To avoid this situation, you can define subpools for exclusive use by individual tasks.

6.25 Any subpool can be used exclusively by a single task or shared by several tasks. Each time that you create a task, you can specify which subpools are to be shared. Unlike other subpools, subpool 0 is shared by a task and its subtask, unless you specify otherwise. When subpool 0 is not shared, the control program creates a new subpool 0 for use by the subtask. As a result, both the task and its subtask can request storage from subpool 0, but both will not receive storage from the same 2048-byte block. When the subtask terminates, its main storage areas in subpool 0 are released; since no other tasks share this subpool, complete 2048-byte blocks are made available for reallocation.

6.26 When there is a need to share subpool 0, you can define other subpools for exclusive use by individual tasks. When you first request storage from a subpool other than subpool 0, the control program assigns a new 2048-byte block to that subpool, and allocates storage from that block. The task that is then active is assigned ownership of the subpool and, therefore, of the block. When additional requests are made by the same task for the same subpool, the requests are satisfied by allocating areas from that block and as many additional blocks as are required. If another task is active when a request is made with the same subpool number, the control program assigns a new block to a new subpool, allocates storage from the new block, and assigns ownership of the new subpool to the second task.

6.27 A task can specify subpools numbered from 0 to 127. FREEMAIN macro instructions can be issued to release any subpool except subpool 0, thus releasing complete 2048-byte blocks. When a task terminates, its unshared subpools are released automatically.

6.28 Owning and Sharing: A subpool is initially owned by the task that was active when the subpool was created. The subpool can be shared with other tasks, and ownership of the subpool can be assigned to other tasks. Two macro instructions are used in the handling of subpools: the GETMAIN macro instruction and the ATTACH macro instruction. In the GETMAIN macro instruction, the SP operand can be written to request storage from subpools 0 to 127; if this operand is omitted, subpool 0 is assumed. The operands that deal with subpools in the ATTACH macro instruction are:

- GSPV and GSPL, which give ownership of one or more subpools (other than subpool 0) to the task being created.
- SHSPV and SHSPL, which share ownership of one or more subpools (other than subpool 0) with the new subtask.
- SZERO, which determines whether subpool 0 is shared with the subtask.

All of these operands are optional. If they are omitted, no subpools are given to the subtask, and only subpool 0 is shared.

6.29 Creating a Subpool: A new subpool is created whenever any of the operands described above is written in an ATTACH or a GETMAIN macro instruction, and that operand specifies a subpool which is not currently owned by or shared with the active task. If one of the ATTACH macro instruction operands causes the subpool to be created, the subpool number is entered in the list of subpools owned by the task, but no blocks are assigned and no storage is actually allocated. If a GETMAIN macro

instruction results in the creation of a subpool, the subpool number is assigned to one or more 2048-byte blocks, and the requested storage is allocated to the active task. In either case, ownership of the subpool belongs to the active task; if the subpool is created because of an ATTACH macro instruction, ownership is transferred or retained depending on the operand used.

6.30 Transferring Ownership: An owning task gives ownership of a subpool to a direct subtask by using the GSPV or GSPL operands in the ATTACH macro instruction issued when that subtask is created. Ownership of a subpool can be given to any subtask of any task, regardless of the control level of the two tasks involved and regardless of how ownership was obtained. A subpool cannot be shared with one or more subtasks and then transferred to another subtask, however; an attempt to do this results in abnormal termination of the active task. Ownership of a subpool can only be transferred if the active task has ownership; if the active task is sharing the subpool and an attempt is made to pass ownership to a subtask, the subtask receives shared control and the originating task

relinquishes the subpool. Once ownership is transferred to a subtask or relinquished, any subsequent use of that subpool number by the originating task results in the creation of a new subpool. When a task that has ownership of one or more subpools terminates, all of the main storage areas in those subpools are released. Therefore, the task with ownership of a subpool should not terminate until all tasks or subtasks sharing the subpool have completed their use of the subpool.

6.31 Sharing a Subpool: Shared use of a subpool can be given to a direct subtask of any task with ownership or shared control of the subpool. Shared use is given by specifying the SHSPV and SHSPL operands in the ATTACH macro instruction issued when the subtask is created. Any task with ownership or shared control of the subpool can add to or reduce the size of the subpool through the use of GETMAIN and FREEMAIN macro instructions. When a task that has shared control of the subpool terminates, the subpool is not affected.

SUBPOOLS IN TASK COMMUNICATION

6.32 The advantage of subpools in main storage management is that, by assigning separate subpools to separate subtasks, the breakdown of main storage into small fragments is reduced. An additional benefit from the use of subpools can be realized in task communication. A subpool can be created for an originating task and all parameters to be passed to the subtask placed in the subpool. When the subtask is created, the ownership of the subpool can be passed to the subtask. After all parameters have been acquired by the subtask, a FREEMAIN macro instruction can be issued, under control of the subtask, to release the subpool main storage areas. In a similar manner, a second subpool can be created for the originating task, to be used as an answer area in the performance of the subtask. When the subtask is created, the subpool ownership would be shared with the subtask. Before the subtask is terminated, all parameters to be passed to the originating task are placed in the subpool area; when the subtask is terminated, the subpool is not released, and the originating task can acquire the parameters. After all parameters have been acquired for the originating task, a FREEMAIN macro instruction again makes the area available for reuse.

IMPLICIT REQUESTS

6.33 You make an implicit request for main storage every time you issue a LINK, LOAD, ATTACH, or XCTL macro instruction. In addition, you make an implicit request for main storage when you issue an OPEN macro instruction for a data set. The data management routines required to process the data set must be in main storage; the main storage areas used as buffers may also be allocated. When you make an implicit request for more main storage than is available, the active task is abnormally terminated. This section discusses some of the techniques you can use to cut down on the amount of main storage required by a job step, and the assistance given you by the control program.

LOAD MODULE MANAGEMENT

6.34 The discussion of program structures indicates the advantages and disadvantages of each of the three types of program designs; simple, planned overlay, and dynamic. The program structure you selected was based on the complexity of the program and the execution time considerations. Once you have selected the program structure, you should plan efficient use of the main storage area that will be assigned to your job step. Note that main storage is assigned in 2048-byte blocks for implicit requests made in an operating system with MVT. The size of your load modules should be planned to take advantage of this method of allo-

cation. The maximum size load module that can be brought into main storage is 524,248 bytes in an operating system with MFT.

REENTERABLE LOAD MODULES

6.35 A reenterable load module is designed so that it does not in any way modify itself during execution. It is "read-only". The advantage of a reenterable load module is most apparent in an operating system with MVT or MFT with subtasking; only one copy of the load module is brought into main storage to satisfy the requirements of any number of tasks in a job step. This means that even though there are six tasks in the job step and each task concurrently requires the load module, the only main storage area requirement is for an area large enough to hold one copy of the load module (plus a few bytes for control blocks). The same main storage requirement would apply if the load module were serially reusable; however, the load module could not be used by more than one task at a time.

6.36 An additional benefit of a reenterable load module occurs when the module is placed in the link pack area. In this case not only is time saved because no loading must be performed, but in addition no main storage area assigned to the job step is required to hold the load module. A link pack area exists only in an operating system with MVT. The contents are established when the operating system is generated and when the operator performs the initial program loading procedure. Any reenterable load module from the link library may be placed in the link pack area. Many of the frequently used data management routines are also placed in the link pack area. If any of your reenterable load modules are used frequently or are used by many jobs, it may save considerable time and space to have those load modules placed in the link pack area.

6.37 Because a reenterable module does not modify itself, a damaged copy of that module can be overlaid with a new copy. Thus reenterable modules offer greater reliability than nonreenterable modules. When a module is designated reenterable (or "refreshable"), the Machine-Check Handler on the Models 65, 85, 155, and 165 automatically loads a fresh copy of that module if it is damaged.

6.38 You can designate a module as refreshable without also designating it as reenterable. However, the module must actually be reenterable in its design, because it must not modify itself during execution.

REENTERABLE MACRO INSTRUCTIONS

6.39 All of the macro instructions described in the macro instructions section can be written in reenterable form. From the standpoint of reenterability, these macro instructions are classified as one of two types: macro instructions which pass parameters in registers 1 and 0, and macro instructions which pass parameters in a list. The use of the macro instructions which pass parameters in registers presents little problem in a reenterable program; when the macro instruction is coded, the required operand values should be contained in registers. For example, the POINT macro instruction requires that the dcb address and block address be coded as follows:

[symbol]	POINT	dcb address,block address

One method of coding a reenterable program would be to require that both of these addresses refer to a portion of main storage allocated to the active task through the use of a GETMAIN macro instruction. The

addresses would change for each use of the load module. Therefore, you would load one of general registers 2-12 with the address, and designate the appropriate registers when you code the macro instruction. If register 4 contained the dcb address and register 6 contained the block address, the POINT macro instruction would be written as follows: POINT (4),(6).

6.40 The macro instructions which pass parameters in a list require the use of special forms of the macro instruction when used in a reenterable program. The expansion of the standard form of these macro instructions results in an in-line parameter list and executable instructions required to branch around the list, to load the address of the list, and to pass control to the required control program routine. The expansions of the list and execute forms of the macro instruction simply divide the functions provided in the standard form expansion: the list form provides only the parameter list, and the execute form provides executable instructions to modify the list and pass control. You provide the instructions to load the address of the list into a register.

6.41 The list and execute forms of a macro instruction are used in conjunction to provide the same services available from the standard form of the macro instruction. The advantages of using list and execute forms are as follows:

- Any operands which remain constant in every use of the macro instruction can be coded in the list form. These operands can then be omitted in each of the execute forms of the macro instruction which use the list. This can save appreciable coding time and main storage area when you use a macro instruction many times. (Any exceptions to this rule are listed in the description of the execute form of the applicable macro instruction.)

- The execute form of the macro instruction can modify any of the operands previously designated. (Again, there are exceptions to this rule.)

- The list used by the execute form of the macro instruction can be located in a portion of main storage assigned to the task through the use of the GETMAIN macro instruction. This ensures that the program remains reenterable.

6.42 Figure 56 shows the use of the list and execute forms of a DEQ macro instruction in a reenterable program. The length of the list constructed by the list form of the macro instruction is obtained by subtracting two symbolic addresses; main storage is allocated and the list is moved into the allocated area. The execute form of the DEQ macro instruction does not modify any of the operands in the list form. The list had to be moved to allocated storage because the control program can store a return code in the list when RET=HAVE is coded. Note that the code in the routine labeled MOVERTN is valid for lengths up to 255 bytes only. Some macro instructions do produce lists greater than 255 bytes when many operands are coded (for example, OPEN and CLOSE with many data control blocks, or ENQ and DEQ with many resources), so in actual practice a length check should be made.

NONREENTERABLE LOAD MODULES

6.43 The use of reenterable load modules does not automatically conserve main storage; in many applications it will actually prove wasteful. If a load module is not used in many jobs and if it is not employed by more than one task in a job step, there is no reason to make the load module reenterable. The allocation of main storage for the purpose of moving code from the load module to the allocated area is a waste of both time and main storage when only one task requires the use of the load module.

```
          ...
          LA        3,MACNAME   Load address of list form
          LA        5,NSIADDR   Load address of end of list
          SR        5,3         Length to be moved in register 5
          BAL       14,MOVERTN  Go to routine to move list
          DEQ       ,MF=(E,(4)) Release allocated resource
          ...
* The MOVERTN allocates storage from subpool 0 and moves up to 255
* bytes into the allocated area.  Register 3 is from address,
* register 5 is length.  Area address returned in register 4.

MOVERTN   GETMAIN   R,LV=(5),   Allocate main storage for list
                    HIARCHY=1   In IBM 2361 Core Storage
          LR        4,1         Address of area in register 4
          BCTR      5,0         Subtract 1 from area length
          EX        5,MOVEINST  Move list to allocated area
          BR        14          Return
MOVEINST  MVC       0(1,4),0(3)
          ...
MACNAME   DEQ       (NAME1,NAME2,8,SYSTEM),RET=HAVE,MF=L
NSIADDR   ...       ...
NAME1     DC        CL8'MAJOR'
NAME2     DC        CL8'MINOR'
```

Figure 56. Using the list and the execute forms of the DEQ macro instruction

6.44 You may remember that, in an operating system with MVT, the area occupied by a reenterable or serially reusable load module is not released automatically when the module returns control to the control program. (Refer to "How Control is Returned" in the discussion of "Passing Control in a Dynamic Structure.") In anticipation of future use, the used copy of the module is retained intact for as long as possible; its area is available to fill both implicit and explicit requests for storage, but only after all other available storage has been allocated. If copies of several modules are retained when they are not needed, available storage may be fragmented as first the areas between the modules are allocated, and then the module areas themselves.

6.45 To prevent this fragmentation, you should not make a load module reenterable or serially reusable if reusability is not really important to the logic of your program. Of course, if reusability is important, you can issue a LOAD macro instruction to load a reusable module, and later issue a DELETE macro instruction to release its area. If reusability is not important, but you need to execute a module that has been made reusable, you can make the module temporarily nonreusable by bringing its directory entry into storage, modifying the contents of the entry, and using the entry to refer to the module. After issuing a BLDL macro instruction to build a list containing the directory entry, you need only set the first two bits of the twenty-third byte in the entry to zero; the module will then be treated as nonreusable when given control by a LINK, ATTACH, or XCTL macro instruction with a DE operand that points to the entry. To set the appropriate bits to zero, you can use an AND-immediate instruction like the following, which could be placed after the BLDL macro instruction in Example 18:

```
NI NAMEADDR+22,B'00111111'
```

This instruction ensures the nonreusability of the module to which NAMEADDR refers.

6.46 One method of conserving main storage when reusability is not a consideration is to use a planned overlay structure. A complete description of the planned overlay structure is contained in the Linkage Editor and Loader manual. Briefly, in a planned overlay structure only portions of the load modules are brought into main storage at a time; when a portion of the load module not in main storage is required, it is loaded in the area occupied by existing portions of the load module. While the use of an overlay structure requires more planning on your part to determine all the portions of a load module required at any one time, it can result in a considerable saving of storage. A well planned overlay structure can result in a savings of 50 percent or more over bringing the entire load module into main storage at once. This does increase the amount of time spent in bringing in portions of the load module, however.

6.47 It is also possible for you to use an overlay type of approach in the design of your load module without using the linkage editor by reusing the areas containing completed routines within a load module. For example, if your load module consists of three control sections of 2000 bytes each which are always executed sequentially, as soon as control is passed to the second control section you have 2000 bytes (the size of the first control section) available to use as a data area. If you reuse this area, you can save up to 2000 bytes of additional main storage which would otherwise be allocated using DS instructions or GETMAIN macro instructions.

RELEASING MAIN STORAGE

6.48 As indicated in Program Management, the control program establishes two responsibility counts for every load module brought into main storage in response to your requests for that load module. The responsibility counts are lowered as follows:

- If the load module was requested in a LOAD macro instruction, that responsibility count is lowered using a DELETE macro instruction.
- If the load module was requested in a LINK, ATTACH, or XCTL macro instruction, that responsibility count is lowered using an XCTL macro instruction or by returning control to the control program.
- When a task is terminated, the responsibility counts are lowered by the number of requests for the load module made in LINK, LOAD, ATTACH, and XCTL macro instructions during the performance of that task, minus the number of deletions indicated above.

6.49 Except for those modules contained in the link pack area, the main storage area occupied by a load module is available for reuse when the responsibility counts reach zero. When you plan your program, you can design the load modules to give you the best trade-off between execution time and efficient main storage use. Naturally, if you will use a load module many times in the course of a job step, you will issue a LOAD macro instruction to bring it into main storage, and you will not issue a DELETE macro instruction until all uses of the load module have completed. In this case it is better to have the load module in main storage all the time than to bring it in every time you require it. Conversely, if a load module is used only once during the job step, or if its uses are widely separated, it will conserve main storage if you issue a LINK macro instruction to load the module and issue an XCTL from the module (or return control to the control program) when it has completed.

6.50 There is a minor problem involved in the deletion of load modules containing data control blocks. An OPEN macro instruction must be

issued before the data control block is used, and a CLOSE macro instruction issued after the use is finished. If you do not issue a CLOSE macro instruction for the data control block, the control program will issue one for you when the task is terminated. However, if the load module containing the data control block has been removed from main storage, the attempt to issue the CLOSE macro instruction will cause abnormal termination of the task. You must either issue the CLOSE macro instruction yourself before deleting the load module, or ensure that the data control block is still in main storage when the task is terminated.

STORAGE HIERARCHIES

6.51 Main storage may be expanded by including IBM 2361 Core Storage in the system (excluding the Model 65 Multiprocessing System). Main Storage Hierarchy Support for IBM 2361 Models 1 and 2 divides main storage into two distinct areas known as hierarchies. In systems incorporating both processor storage and 2361 Core Storage, hierarchy 0 is assigned to processor storage and hierarchy 1 is assigned to 2361 Core Storage. The first address in 2361 storage is one higher than the highest address in processor storage.

6.52 In MFT, storage hierarchy structures are established for partitions during System Generation, according to user specifications. These defined structures may be redefined by the operator any time after system initialization. A partition may be contained entirely within one hierarchy, or may consist of one partition segment in hierarchy 0, and another in hierarchy 1. If 2361 Core Storage is not on line at system initialization, only partition segments defined in hierarchy 0 are reserved, and only the amount of storage specified for the hierarchy 0 segment of the partition is allocated.

6.53 In MVT, a region storage hierarchy structure is established via the REGION parameter in the job control language JOB and EXEC statements. If 2361 Core Storage is not on line at system initialization in an MVT system with Main Storage Hierarchy Support, and a region segment structure is specified in both hierarchies in the JOB or EXEC statement, the control program will define two separate region segments (not necessarily contiguous) in processor storage. The region segments will be addressable as hierarchy 0 and hierarchy 1, each hierarchy being assigned its respective size as indicated in the REGION parameter of the JOB or EXEC statement.

6.54 Hierarchies 0 and 1 may be specified by the hierarchy parameter (HIARCHY=) in the ATTACH, DCB, GEMAIN, GETPOOL, LINK, LOAD, and XCTL macro instructions. If the hierarchy parameter is omitted, requested storage, if available, is obtained from processor storage.

6.55 If a partition or a region is defined entirely in one hierarchy, all requests for storage will be directed to that hierarchy regardless of the hierarchy designated by the HIARCHY= parameter.

6.56 Figure 54 shows two GETMAIN requests for storage from hierarchy 0. Figure 56 illustrates the use of a GETMAIN macro instruction in requesting storage from hierarchy 1. Requirements for writing macro instructions with the hierarchy parameter are described in the macro instructions section.

6.57 In using Main Storage Hierarchy support on a Model 50, use caution in directing programs containing CCWs for direct access devices to be loaded into hierarchy 1. (Under MFT, this includes readers and writers.) If this is disregarded, overrun will occur which will degrade the performance or result in an unrecoverable I/O error.

7.1 The checkpoint/restart description has been deleted from this publication. The Advanced Checkpoint/Restart manual contains complete information for using the facility. The topic, "Using the Restart Facilities" in the Job Control Language Reference manual contains information on restart.

SECTION II: MACRO INSTRUCTIONS

INTRODUCTION

8.1 A set of macro instructions is provided by IBM for communicating service requests to the control program. These 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.

8.2 The processing of the macro instruction by the assembler program results in a macro expansion, generally consisting of data and executable instructions in the form of assembler language statements. The data fields are the parameters to be passed to the requested control program routine; the executable instructions generally consist of a branch around the data, instructions to load registers, and either a branch instruction or a supervisor call (SVC) to give control to the proper program. The exact macro expansion appears as part of the assembler program output listing.

8.3 A listing of a macro definition from the MACLIB can be obtained by using the utility IEBPTPCH, which is described in the *Utilities* publication.

OPERATING SYSTEM CONFIGURATIONS AND OPTIONS

8.4 The operation of some macro instructions depends on control program options. For these macro instructions, either separate descriptions are provided or the differences are listed within a single description. If no differences are explicitly listed, none exist.

8.5 A brief description of the MFT and MVT configurations of the operating system, along with control program options available in each configuration, is given in Figure 57. This table does not attempt to cover all of the options available in the operating system; it only summarizes the options that affect the material in this manual.

	MFT	MVT
Brief Description	Priority Scheduler, one (or, optionally, more than one) task per job step, 1 to 15 jobs processed concurrently	Priority Scheduler, one or more tasks per job step, 1 to 15 jobs processed concurrently
Attach	Optional	Standard
Identify	Optional	Standard
Timer	Optional	Standard
Interval Timer	Optional	Standard
Multiple Console Support	Optional	Optional
Time Sharing	Not Available	Optional

Figure 57. Summary of characteristics and available options

CODING AIDS

8.6 The symbols [ ], { }, and ,... are used in this publication to help define the macro instructions. THESE SYMBOLS ARE NOT CODED; they act only to indicate how a macro instruction may be written. The specific meanings of these symbols are given at the bottom of each page on which they are used; their general definitions are given below:

[ ] indicates optional operands. The operand enclosed in the brackets (for example, [VL]) may or may not be coded, depending on whether or not the associated option is desired. If the operand is not coded, any default value is indicated by an underline. If more than one item is enclosed in brackets (for example, [STEP / SYSTEM]), one or none of the items may be coded.

{ } indicates that a choice must be made. One of the operands from the vertical stack within braces (for example, {YES / NO}) must be coded, depending on which of the associated services is desired.

,... indicates that more than one set of operands may be designated in the same macro instruction.

WRITING THE MACRO INSTRUCTIONS

8.7 The system macro instructions are written in the assembler language, and, as such, are subject to the rules contained in the Assembler Language publication. System macro instructions, like all assembler language instructions, are written in the following format:

Name	Operation	Operands	Comments
symbol or blank	Macro name	None, or one or more operands separated by commas	

8.8 The operands are used to specify the services and options to be performed, and are written according to the following general rules:

- If the selected operand is written in all capital letters (for example, STEP, DUMP, RET=USE), code the operand exactly as shown.
- If the selected operand is written in lower case letters, substitute the indicated value, address, or name.
- If the selected operand is a combination of capital and lower case letters separated by an equal sign (for example, EP=entry point name), code the capital letters and equal sign as shown, then make the indicated substitution.
- Commas and parentheses are coded exactly as shown, except that a comma following the last operand coded by the programmer should be omitted. The use of commas and parentheses is indicated by brackets and braces, exactly as operands.

8.9 When substitution is required, the method of specifying the operand depends on the requirements of the control program. The description of each operand in the standard form indicates how the operand should be coded, in addition to what is to be coded. The descriptions of the list and execute forms indicate only how the operands should be coded. Figure 70 summarizes how each operand in each form is to be coded. The classifications are as follows:

Sym
: is any symbol valid in the assembler language.

Dec Dig
: are any decimal digits up to the value indicated in the associated macro instruction description. If both SYM and DEC DIG are checked, an absolute expression is also allowed.

REGISTER
: is always coded within parentheses, as follows:
  - (2-12) - one of general registers 2 through 12, previously loaded with the right-adjusted value or address specified in the associated macro instruction description. The unused high-order bits must be set to zero. The register may be designated symbolically or with an absolute expression.
  - (1) - general register 1, previously loaded as indicated above. The register can be designated only as (1).
  - (0) - general register 0, previously loaded as indicated above. The register can be designated only as (0).

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

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

CONTINUATION LINES

8.10 The operand field of a macro instruction can be continued on one or more additional lines as follows:

1. Enter a continuation character (not blank, and not part of the operand coding) in column 72 of the line.

2. Continue the operand field on the next line, starting in column 16. All columns to the left of column 16 must be blank.

8.11 The operand field being continued can be coded in one of two ways. The operand field can be coded through column 71, with no blanks, and continued in column 16 of the next line, or the operand field can be truncated by a comma, where a comma normally falls, with at least one blank before column 71, and then continued in column 16 of the next line. An example of each method is shown in Figure 58.

8.12 Additional information on the continuation of any assembler language macro instruction is provided in the Assembler Language publication.

Name	Operation	Operand	Comments	
NAME1	OP1	OPERAND1,OPERAND2,OPERAND3,OPERAND4,OPERA		X
		ND5,OPERAND6	THIS IS ONE WAY	
NAME2	OP2	OPERAND1,OPERAND2,	THIS IS	X
		OPERAND3,	ANOTHER WAY	X
		OPERAND4		

Figure 58. Continuation Coding

ADDITIONAL MACRO INSTRUCTIONS

8.13 The following macro instructions are described in the MFT Guide and the MVT Guide:

ATLAS	PURGE
CATALOG	QEDIT
CIRB	RESERVE
CVT	RESTORE
EOV	SMFWTM
LABEL	SYNCH

8.14 The following macro instructions are described in Data Management for System Programmers:

CAMLIST	OBTAIN
DEVTYPE	OPEN (TYPE=J)
EXCP	PROTECT
IECDSECT	RDJFCB
IEFJFCBN	RENAME
IEFUCBOB	SCRATCH
INDEX	XDAP
LOCATE	

8.15 The following macro instructions are described in Data Management Macro Instructions:

BLDL	CNTRL	FREEBUF	NOTE	READ	SYNADAF
BSP	DCB	FREEDBUF	OPEN	RELEX	SYNADRLF
BUILD	DCBD	FREEPOOL	POINT	RELSE	WRITE
BUILDRCD	ESETL	GET	PRTOV	SETL	TRUNC
CHECK	FEOV	GETBUF	PUT	SETPRT	XLATE
CLOSE	FIND	GETPOOL	PUTX	STOW	

8.16 The following macro instructions are described in the TSO Guide to Writing a Terminal Monitor Program or a Command Processor:

STATTN	STCOM
STATUS	STSIZE
STAUTOLN	STTIMEOU
STAX	TCLEARQ
STBREAK	TGET
STCC	TPUT
STCLEAR	

STANDARD, LIST, AND EXECUTE FORMS

9.1 The standard, list, and execute forms of each macro instruction (where applicable) are grouped for ease of reference. The standard form of a macro instruction causes operand specifications (if any) to be indicated by parameters passed in registers or in an in-line parameter list. With the exception of the CALL macro instruction, the standard form also causes control to be passed to a control program routine to perform the requested function. The option of using an out-of-line parameter list allows the use of these macro instructions in a reenterable program. The option is requested through the list and execute forms

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

9.3 The execute form of the macro instruction is used 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, where possible, the parameter lists and to pass control to the required program. Only the ATTACH, LINK, and XCTL macro instructions use two parameter lists; a problem program list, resulting from the address parameter and VL operands, and a control program list, resulting from the remaining operands. The control program list is required and the problem program list is optional in these macro instructions.

9.4 An operand value specified in the list form of a macro instruction remains in effect until changed by the respecification of the operand in an execute form of the macro instruction. Any exceptions to this rule are indicated in the description of the execute forms of the macro instruction. This rule does not apply to suboperands of operands that are modified such as the PURGE and ASYNCH suboperands of the STAE operand in the ATTACH macro instruction. Unless otherwise specified, default values are assigned only when omitted from the list form, not the execute form, of a macro instruction.

9.5 The SNAP macro instruction can result in a variable length parameter list. The length of the parameter list generated by the list form of the macro instruction must be equal to the maximum length list required by any execute form which refers to the list. The maximum length list can be constructed in one of three ways:

- Code the parameters required for the maximum length execute form in the list form.
- Provide a DS instruction immediately following the list form to allow for the maximum length parameter list.
- Acquire a maximum length list by using commas in the list form to indicate the maximum number of parameters. For example, the STORAGE operand of the SNAP macro instruction could be coded as STORAGE=(,,,,,,,,,) to allow for five pairs of addresses. The actual address would be provided in the execute form.

9.6 The description and definition of each macro instruction and the allowable methods of coding each operand are provided with the standard form. The allowable methods of coding and descriptions of operands that are unique with the list and execute forms are provided with the list and execute forms.

ABEND -- Abnormally Terminate a Job Step (MFT Without Subtasking)

10.1 The ABEND macro instruction causes the current job step to be abnormally terminated. The specified completion code is recorded on the system output device and a dump is optionally provided. All main storage areas assigned to the job step are released, and the remaining job steps in the job are either skipped or executed as specified in their job control statements.

10.2 The ABEND macro instruction is written as shown in the format description below. The operand in the shaded area is used only in an operating system with MVT or MFT with subtasking; it is ignored if coded in an operating system with MFT without subtasking. The operands in the nonshaded area can be coded with any configuration of the operating system.

[symbol]	ABEND	completion code, [DUMP] [,STEP]

10.3 completion code — Sym, Dec Dig, (1-12)

is a maximum of 4095. This number is labeled user code on the system output device. Using a value greater than 4095 will cause unpredictable user and/or system completion codes.

10.4 DUMP

is written as shown. It is used to request a dump of all main storage areas assigned to the job step and the control blocks belonging to the job step. The trace table and nucleus are also recorded if a //SYSABEND DD statement is provided. Sample abnormal termination dumps are contained in the Programmer's Guide to Debugging. If this operand is omitted, no dump is provided. A //SYSABEND or //SYSUDUMP DD statement must be provided to obtain the dump. If the DD statement is omitted or its specifications destroyed, an indicative dump is provided on the system output device.

ABEND

ABEND -- Abnormally Terminate a Task (MVT, MFT With Subtasking)

11.1 The ABEND macro instruction causes the control program to abnormally terminate the active task and all the subtasks of the active task. The ABEND macro instruction can request a dump of all main storage areas and control blocks pertaining to the tasks being abnormally terminated, and can specify that the entire job step is to be abnormally terminated. If the job step task is abnormally terminated or if the ABEND macro instruction specified job step termination, the completion code is recorded on the system output device, and the remaining job steps in the job are either skipped or executed as specified in their job control statements.

11.2 If the job step is not to be terminated, the following action is taken:

- The task that was active when the ABEND macro instruction was issued is terminated, along with all of the subtasks of that active task.
- The completion code is posted as indicated in the completion code operand description.
- One end-of-task exit routine is selected to be given control. This is the exit routine specified in the ATTACH macro instruction that created the task that issued the ABEND macro instruction. The exit routine will be given control when the originating task of the task for which the ABEND macro instruction was issued becomes active. None of the end-of-task exit routines specified for any subtasks of the task for which the ABEND macro instruction was issued are given control.

11.3 The ABEND macro instruction is written as follows:

[symbol]	ABEND	completion code[,DUMP][,STEP]

11.4 completion code — Sym, Dec Dig, (1-12)

is a maximum of 4095. Using a value greater than 4095 will cause unpredictable user and/or system completion codes. If the job step is to be terminated, the completion code is recorded as user code on the system output device. If the job step is not to be terminated, the completion code is placed in the task control block of the active task, and in the event control block specified in the ECB operand of the ATTACH macro instruction issued to create the active task.

11.5 DUMP

is written as shown. It is used to request a dump of all main storage areas assigned to the task and all the control blocks pertaining to the task. If a //SYSABEND DD statement is provided, the nucleus is also recorded. A sample abnormal termination dump is contained in Programmer's Guide to Debugging. A separate dump is provided for each of the tasks being terminated as a result of the ABEND macro instruction. In addition, a dump of the control blocks and save areas is provided for each of the higher level tasks that are direct predecessors of the task being terminated. A //SYSABEND or a //SYSUDUMP DD statement must be provided; if it is not, the DUMP operand is ignored. If the operand is omitted or if insufficient main storage area is available in the region for the abnormal termination to be performed, no dump is provided.

11.6 STEP

is written as shown. It indicates that the entire job step of the active task is to be abnormally terminated.

ATTACH -- Pass Control to a Program in Another Load Module (MFT Without Subtasking)

12.1 The ATTACH macro instruction causes control to be passed to a program at a specified entry point; the load module containing the program is brought into main storage if a usable copy is not available. (Refer to Section I for a discussion of the use of an existing copy of a load module.) The linkage relationship established is the same as that created by a BAL instruction; control is returned to the instruction following the ATTACH macro instruction after execution of the called program. The entry point name that is specified must be a member name or an alias in a directory of a partitioned data set, or have been specified in an IDENTIFY macro instruction.

12.2 The problem program optionally can provide a parameter list to be passed to the called program, can provide an event control block to receive the completion code from the called program, and can specify an exit routine to be given control when the called program terminates. If the called program terminates abnormally, or if the specified entry point cannot be located, the job step is abnormally terminated.

12.3 The standard form of the ATTACH macro instruction is written as shown in the format description below. All the operands in the shaded area of this format description can be used in an operating system with MVT. The LPMOD and DPMOD operands in the shaded area can also be used in an operating system with MFT with subtasking. However, all the operands in the shaded area are ignored if coded in an operating system with MFT without subtasking. The operands in the nonshaded area can be coded with any configuration of the operating system.

```
[symbol]   ATTACH   {EP=symbol                }
                    {EPLOC=address of name    }
                    {DE=address of list entry }

                    [,DCB=dcb address]
                    [,PARAM=(addresses)[,VL=1]]
                    [,ECB=ecb address] [,ETXR=exit routine address]
                    [,HIARCHY=number]

                    --- shaded area ---
                    [,LPMOD=number] [,DPMOD=number]
                    [,GSPV=number           ] [,SHSPV=number          ]
                    [,GSPL=address of list  ] [,SHSPL=address of list ]

                    [,SZERO={YES}]
                            {NO }

                    [,STAI=(exit address[,parameter list address])]

                    [,PURGE={QUIESCE}] [,ASYNCH={YES}]
                            {HALT   }           {NO }
                            {NONE   }

                    [,TASKLIB=dcb address]
```

12.4 EP= Sym

is the entry point name in the load module to be given control.

12.5 EPLOC= A-type, (2-12)

is the main storage address of an entry point name. The name must be padded to the right with blanks to eight bytes, if necessary.

ATTACH

12.6 DE= A-type (2-12)
is the address of the name field of a list entry for the entry point name. The list entry is constructed by a BLDL macro instruction. The DCB operand must indicate the same data control block used in the BLDL macro instruction.

12.7 DCB= A-type (2-12)
is the address of the data control block for the partitioned data set containing the entry point name. The address of the data control blocks for the link and job libraries is designated by specifying an address of zero or by omitting the DCB operand.

12.8 PARAM= A-type (2-12)
is one or more address parameters, separated by commas, to be passed to the called program. Each address is expanded in line to a full word on a full word boundary, in the order designated. Register 1 contains the address of the first parameter when the program is given control. (If this operand is omitted, register 1 is not altered.)

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

12.10 ECB= A-type (2-12)
is the address of a fullword on a fullword boundary. The control program indicates normal termination of the called program by placing the return code from the called program into the low-order three bytes of the fullword.

12.11 ETXR= A-type (2-12)
is the address of an exit routine to be given control after normal termination of the called program. The contents of the registers when the exit routine is given control are as follows:

Register	Contents
0	Control program information.
1	Set to zero.
2-12	Unpredictable.
13	Address of the save area provided by the control program.
14	Return address (to control program).
15	Address of exit routine.

12.12 The exit routine must be in main storage when it is required and must return control to the control program.

12.13 HIARCHY= Dec Dig
specifies the storage hierarchy (0 or 1) into which the load module is to be loaded when a usable copy is not already available in main storage. If the HIARCHY parameter is not specified, loading will take place according to the hierarchy specified at Link Edit time. If the HIARCHY request can be satisfied, it will override any hierarchy assignments made during Link Edit time. If the HIARCHY request cannot be satisfied, the ATTACH macro instruction return an error code of 04 in register 15. The HIARCHY parameter is ignored in an operating system that does not have main storage hierarchy support.

ATTACH -- Create a New Task (MFT With Subtasking)

13.1 The ATTACH macro instruction causes the control program to create a new task that will execute asynchronously in the same partition as the calling task. The entry point name that is specified must be a member name or an alias in a directory of a partitioned data set, or have been specified in an IDENTIFY macro instruction. If the specified entry point cannot be located, the new subtask is abnormally terminated.

13.2 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 dispatching priority determines whether the new task participates in time slicing (only if time slicing is included in the system).

13.3 The load module containing the program to be given control is brought into main storage if a usable copy is not available in main storage. (For further discussions of the use of an existing copy of a load module, refer to Section I.)

13.4 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.

13.5 If the ECB or ETXR operands are coded, a DETACH macro instruction must be issued to remove the subtask from the system before the program that issued the ATTACH macro instruction terminates. If the ECB or ETXR operands are not coded, the subtask will automatically be removed from the system upon completion of its processing.

13.6 Notes:

- The ATTACH macro instruction cannot be issued in a STAE exit routine.
- The program issuing the ATTACH macro instruction must not terminate before all of its subtasks have terminated.
- Concatenated SYSIN/SYSOUT Unit Record data sets cannot be processed (I/O requests and close requests cannot be issued) from a subtask if the subtask did not originally open the SYSIN/SYSOUT data set.

13.7 The standard form of the ATTACH macro instruction is written as follows. The operands in the shaded area of the format description are used only in MVT; they are ignored if coded in any other configuration of the operating system.

13.8

ATTACH

```
[symbol]   ATTACH   {EP=symbol                 }
                    {EPLOC=address of name     }
                    {DE=address of list entry  }

                    [,DCB=dcb address]
                    [,PARAM=(addresses)[,VL=1]]
                    [,ECB=ecb address][,ETXR=exit routine address]
                    [,HIARCHY=number]
                    [,LPMOD=number][,DPMOD=number]

                    [,GSPV=number          ] [,SHSPV=number          ]
                    [,GSPL=address of list ] [,SHSPL=address of list ]

                    ,SZERO={YES}
                           {NO }

                    [,STAI=(exit address[,parameter list address])]

                    [,PURGE={QUIESCE}][,ASYNCH={YES}]
                    [       {HALT   }]        {NO }
                    [       {NONE   }]

                    [,TASKLIB=dcb address]
```

13.8 EP= Sym

is the entry point name in the load module to be given control.

13.9 EPLOC= A-type, (2-12)

is the main storage address of the entry point name. The name must be padded to the right with blanks to eight bytes, if necessary.

13.10 DE= A-type, (2-12)

is the address of the name field of a list entry for the entry point name. The list entry is constructed using the BLDL macro instruction. The DCB operand must indicate the same data control block used in the BLDL macro instruction. If the module is indicated as being in the job, step, or task library by the Z byte of the BLDL list entry, the ATTACH must be either in the same task as the BLDL or in a task with the same chain of task libraries.

13.11 DCB= A-type, (2-12)

is the address of the data control block for the partitioned data set containing the entry point name. The address of the data control block for either the link or job library is designated by specifying an address of zero or by omitting the DCB operand.

13.12 PARAM= A-type, (2-12)

is one or more address parameters, separated by commas, to be passed to the called program. Each address is expanded in line to a fullword on a fullword boundary, in the order designated. Register 1 contains the address of the first parameter when the program is given control. If this operand is omitted, register 1 is not altered.

13.13 VL=1

is written as shown. It can be designated only if PARAM is designated, and should be used only if the called program can be passed a variable number of parameters. VL=1 causes the high-order bit of the last address parameter to be set to 1; the bit can be checked to find the end of the list.

13.14 ECB= A-type, (2-12)

is the address of an event control block to be used by the control program to indicate the termination of the new 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 operand is coded, a DETACH macro instruction must be issued to remove the subtask from the system after the subtask has been terminated.

13.15 ETXR= A-type, (2-12)

is 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 main storage when required. If the same routine is used for more than one subtask, it must be reenterable. If this operand is coded, a DETACH macro instruction must be issued to remove the subtask from the system after the subtask has been terminated. 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.

13.16 Because it operates as a logical subroutine, the exit routine must return to the control program when its processing is complete.

13.17 HIARCHY= Dec Dig

specifies the storage hierarchy (0 or 1) into which the load module is to be loaded when a usable copy is not already available in main storage. If the HIARCHY parameter is not specified, loading will take place according to the hierarchy specified at Link Edit time. If the HIARCHY request can be satisfied, it will override any hierarchy assignments made at Link Edit time. If the HIARCHY request cannot be satisfied, the ATTACH macro instruction returns an error code of 04 in register 15. The HIARCHY parameter is ignored in an operating system that does not have main storage hierarchy support.

13.18 LPMOD= Sym, dec dig, (2-12)

is the number to be subtracted from the current limit priority of the originating task. The result is the limit priority of the new task. If omitted, the current limit priority of the originating task is assigned as the limit priority of the new task.

13.19 DPMOD= Sym, Dec Dig, (2-12)

is the signed number 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 operand is omitted, the dispatching priority assigned is the smaller of either the new task's limit priority or the originating task's dispatching priority.

ATTACH

ATTACH -- Create a New Task (MVT)

14.1 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 have been specified in an IDENTIFY macro instruction. If the specified entry point cannot be located, the new subtask is abnormally terminated.

14.2 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 dispatching priority determines whether or not the new task participates in time slicing (only in a system that includes the time-slicing option).

14.3 The load module containing the program to be given control is brought into main storage if a usable copy is not available in main 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 the ECB or ETXR operands are coded, a DETACH macro instruction must be issued to remove the subtask from the system before the program that issued the ATTACH macro instruction terminates. If the ECB or ETXR operands are not coded, the subtask will automatically be removed from the system upon completion of its processing. The ATTACH macro instruction can also be used to specify that ownership of main storage 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.

14.4 Notes:

- The ATTACH macro instruction cannot be issued in a STAE exit routine.
- The program issuing the ATTACH macro instruction must not terminate before all of its subtasks have terminated.
- Concatenated SYSIN/SYSOUT Unit Record data sets cannot be processed (I/O requests and close requests cannot be issued) from a subtask if the subtask did not originally open the SYSIN/SYSOUT data set.

14.5 For further discussions of time slicing and the use of an existing copy of a load module, refer to Section I.

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

```
[symbol]  ATTACH  {EP=symbol                  }
                  {EPLOC=address of name      }
                  {DE=address of list entry   }

                  [,DCB=dcb address]
                  [,LPMOD=number] [,DPMOD=number]
                  [,PARAM=(addresses)[,VL=1]]
                  [,ECB=ecb address] [,ETXR=exit routine address]
                  [,HIARCHY=number]

                  [,GSPV=number             ] [,SHSPV=number             ]
                  [,GSPL=address of list    ] [,SHSPL=address of list    ]

                  [,SZERO={YES}]
                          {NO }

                  [,STAI=(exit address[,parameter list address])          ]
                  [                                                       ]
                  [[,PURGE={QUIESCE}] [,ASYNCH= {YES}]                    ]
                  [        {HALT   }           {NO }                      ]
                  [        {NONE   }                                      ]

                  [,TASKLIB=dcb address]
```

(Underlined defaults: YES for SZERO, QUIESCE for PURGE, NO for ASYNCH.)

14.7 EP= Sym

is the entry point name in the load module to be given control.

14.8 EPLOC= A-type, (2-12)

is the main storage address of the entry point name. The name must be padded with blanks to eight bytes, if necessary.

14.9 DE= A-type, (2-12)

is the address of the name field of a list entry for the entry point name. The list entry is constructed using the BLDL macro instruction. The DCB operand must indicate the same data control block used in the BLDL macro instruction. If the module is indicated as being in the job, step, or task library by the Z byte of the BLDL list entry, the ATTACH must be either in the same task as the BLDL or in a task with the same chain of task libraries.

14.10 DCB= A-type, (2-12)

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

If the DCB= operand is omitted or if DCB=0 is specified when the ATTACH macro instruction is issued by the job step task, the data sets referenced by either the STEPLIB or JOBLIB DD statement are first searched for the entry point name. If the entry point name is not found, the link library is searched.

If the DCB= operand is omitted or if DCB=0 is specified when the ATTACH macro instruction is issued by a subtask, the data set(s) associated with one or more data control blocks referenced previous ATTACH macro instructions in the subtasking chain are first searched for the entry point name. If the entry point name is not found, the search is continued as if the ATTACH macro instruction had been issued by the job step task.

14.11 LPMOD= Sym, Dec Dig, (2-12)

is the number to be subtracted from the current limit priority of the originating task. The result is the limit priority of the new task. If omitted, the current limit priority of the originating task is assigned as the limit priority of the new task.

14.12 DPMOD= Sym, Dec Dig, (2-12)
is the signed number 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 operand is omitted, the dispatching priority assigned is the smaller of either the new task's limit priority or the originating task's dispatching priority.

14.13 PARAM= A-type, (2-12)
is one or more address parameters, separated by commas, to be passed to the called program. Each address is expanded in line to a fullword on a fullword boundary, in the order designated. Register 1 contains the address of the first parameter when the program is given control. If this operand is omitted, register 1 is not altered.

14.14 VL=1
is written as shown. It can be designated only if PARAM is designated, and should be used only if the called program can be passed a variable number of parameters. VL=1 causes the high-order bit of the last address parameter to be set to 1; the bit can be checked to find the end of the list.

14.15 ECB= A-type, (2-12)
is the address of an event control block to be used by the control program to indicate the termination of the new 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 operand is coded, a DETACH macro instruction must be issued to remove the subtask from the system after the subtask has been terminated.

14.16 ETXR= A-type, (2-12)
is 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 main storage when required. If the same routine is used for more than one subtask, it must be reenterable. If this operand is coded, a DETACH macro instruction must be issued to remove the subtask from the system after the subtask has been terminated. 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.

14.17 HIARCHY= Dec Dig
specifies the storage hierarchy (0 or 1) into which the load module is to be loaded when a usable copy is not already available in main storage. If the HIARCHY parameter is not specified, loading will take place according to the hierarchy specified at Link Edit time.

If the HIARCHY request can be satisfied, it will override any hierarchy assignments made during Link Edit time. If the HIARCHY request cannot be satisfied, the ATTACH macro instruction returns an error code of 04 in register 15. The HIARCHY operand is ignored in an operating system that does not have main storage hierarchy support.

14.18 GSPV= Sym, Dec Dig, (2-12)

is a main storage subpool number. Ownership of the specified main storage subpool is assigned to the new task. Programs of the originating task can no longer use the associated main storage area.

14.19 GSPL= A-type, (2-12)

is the address of a list of main storage subpool numbers. The first byte of the list contains the number of remaining bytes in the list; each of the following bytes contains a main storage subpool number. Ownership of each of the specified main storage subpools is assigned to the new task. Programs of the originating task can no longer use the associated main storage areas.

14.20 SHSPV= Sym, Dec Dig, (2-12)

is a main storage subpool number. Programs of both the originating task and the new task can use the associated main storage area.

14.21 SHSPL= A-type, (2-12)

is the address of a list of main storage subpool numbers. The first byte of the list contains the number of remaining bytes in the list; each of the following bytes contains a main storage subpool number. Programs of both the originating task and the new task can use the associated main storage areas.

14.22 SZERO=

is used to indicate whether subpool zero is to be shared with the subtask. YES specifies subpool zero will be shared; NO specifies subpool zero will not be shared. YES is assumed if this operand is omitted.

14.23 STAI=

exit address A-type, (2-12)

is the address of the exit routine that receives control when any subtask of the ATTACH-issuing task is scheduled for abnormal termination (ABEND). This exit routine will also receive control if any lower-level subtask of the newly created subtask is scheduled for abnormal termination and does not successfully recover from the error. The routine must be in main storage when the ABEND occurs. (The STAI exit routine performs the same functions for a subtask that the STAE exit routine performs for the issuing task. See Section I for a description of the STAE exit routine.)

parameter list address A-type, (2-12)

is the address of any parameter list that might be required by the STAI exit routine.

14.24 PURGE=

QUIESCE

indicates that all outstanding requests for input/output operations will be saved when the specified STAI exit is taken. Then at the end of the STAI exit routine, the user can code a retry routine to handle the outstanding input/output requests. (See the description of the STAE macro instruction in the MFT Guide or the MVT Guide for

ATTACH

an explanation of the STAE retry routine.) If the PURGE operand is not specified, QUIESCE is assumed.

HALT
indicates that all outstanding requests for input/output operations will not be saved when the STAI exit is taken. A retry routine should not be scheduled if PURGE=HALT is specified.

NONE
indicates that input/output processing is allowed to continue normally when the STAI exit is taken.

Note: If the STAI operand is not specified, the PURGE operand is ignored.

14.25 ASYNCH=

YES
indicates that asynchronous interrupt processing is allowed to interrupt the processing done by the STAI exit routine.

NO
indicates that asynchronous interrupt processing is not allowed to interrupt the processing done by the STAI exit routine. If neither YES nor NO is coded, NO is assumed.

Note: If the STAI operand is not specified, the ASYNCH operand is ignored.

14.26 TASKLIB= A-type, (2-12)
is the address of an opened data control block for the job library. This library, now called a task library, is searched for the entry point name of the module being attached and for the entry point names of subsequent modules accessed by the subtask. If the TASKLIB operand is not specified, the job library whose data control block (DCB) address is found in the attaching task's task control block (TCB) is searched instead. All modules contained in the job library and task libraries for a job step should be uniquely named; if duplicate names appear in these libraries, the search results are unpredictable.

14.27 If the ATTACH macro instruction is executed successfully, control is returned to the user with one of the following return codes in register 15:

Hex Code	Meaning
00	Successful completion of the ATTACH request.
04	The ATTACH macro instruction was issued in a STAE Exit routine. The subtask was not created.
08	Sufficient main storage was not available to schedule the exit routine as specified by the STAI operand. The subtask was not created.
0C	The exit routine or parameter list address specified in the STAI operand was invalid. The subtask was not created.
10	The storage required for the STAI request was not available. The subtask was not created.

ATTACH -- List Form

15.1 Two parameter lists are used in an ATTACH macro instruction: a control program parameter list and an optional problem program parameter list. Only the control program parameter list can be constructed in the list form of the ATTACH macro instruction. 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 the ATTACH macro instruction.

15.2 The description of the standard form of the ATTACH macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the list form only. The operands in the shaded area are used in MVT only; they are ignored if coded with MFT. The LPMOD and DPMOD operands of this format description are used with MVT and MFT with subtasking. In MFT systems without subtasking, these two operands are ignored if coded. The operands in the nonshaded area can be coded with any control program.

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

```
[symbol]   ATTACH   {EP=symbol              }
                    {EPLOC=address of name  }
                    {DE=address of list entry}

                    [,DCB=dcb address]

                    [,ECB=ecb address] [,ETXR=exit routine address]
                    [,HIARCHY=number]

                    [,LPMOD=number] [,DPMOD=number]

                    [,GSPV=number            ] [,SHSPV=number            ]
                    [,GSPL=address of list   ] [,SHSPL=address of list   ]

                    [,SZERO={YES}]
                    [       {NO }]

                    [,STAI=(exit address[,parameter list address])       ]
                    [                                                    ]
                    [ [,PURGE={QUIESCE}] [,ASYNCH= {YES}]                ]
                    [         {HALT   }            {NO }]                ]
                    [         {NONE   }                                  ]
                    [,TASKLIB=dcb address]

                    ,SF=L
```

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

number
is any absolute expression valid in the assembler language.

SF=L
indicates the list form of the ATTACH macro instruction.

ATTACH - E Form

ATTACH -- Execute Form

16.1 Two parameter lists are used in an ATTACH macro instruction: 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 the ATTACH macro instruction. If only one of the parameter lists is remote, operands that require use of the other parameter list cause that list to be constructed in line as part of the macro expansion.

16.2 The description of the standard form of the ATTACH macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the execute form only. The operands in the shaded area are used only with MVT; they are ignored if coded with MFT. The LPMOD and DPMOD operands of this format description are used with MVT and MFT systems with subtasking. In MFT systems without subtasking, these two operands are ignored if coded. The operands in the nonshaded area can be coded with any control program.

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

```
[symbol] | ATTACH | { EP=symbol                }
         |        | { EPLOC=address of name    }
         |        | { DE=address of list entry }
         |        |
         |        | [,DCB=dcb address]
         |        | [,PARAM=(addresses) [,VL=1]]
         |        | [,ECB=ecb address] [,ETXR=exit routine address]
         |        | [,HIARCHY=number]
         |        |
         |        | [,LPMOD=number] [,DPMOD=number]
         |        |
         |        | [,GSPV=number             ] [,SHSPV=number             ]
         |        | [,GSPL=address of list    ] [,SHSPL=address of list    ]
         |        |
         |        | [,SZERO={YES}]
         |        | [       {NO }]
         |        |
         |        | [,STAI=(exit address[,parameter list address])              ]
         |        | [                                                           ]
         |        | [          {QUIESCE}                                        ]
         |        | [[,PURGE={HALT   }] [,ASYNCH={YES}]                         ]
         |        | [          {NONE   }          {NO }                         ]
         |        |
         |        | [,TASKLIB=dcb address]
         |        |
         |        | ( ,MF=(E,{problem program list address})              )
         |        | (        {(1)                         }               )
         |        | (                                                     )
         |        | ( ,SF=(E,{control program list address})              )
         |        | (        {(15)                        }               )
         |        | (                                                     )
         |        | ( ,MF=(E,{address}),SF=(E,{address})                  )
         |        | (        {(1)    }        {(15)   }                   )
```

16.4 address

is any address that is valid in an RX-type instruction, or one of general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

16.5 number

is any absolute expression that is valid in the assembler language, or one of general registers 2 through 12, previously loaded with the indicated value. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

16.6 MF=(E,{problem program list address | (1)})

indicates the execute form of the macro instruction using a remote problem program parameter list. Any control program parameters specified are provided in a control program parameter list expanded in line. If the PARAM operand is also specified, the address parameters will be placed on contiguous fullword boundaries, beginning at the address specified in the MF operand and sequentially overlaying corresponding fullwords in the existing list. The address of the problem program parameter list can be coded as described under "address," or can be loaded into register 1, in which case MF=(E,(1)) should be coded.

16.7 SF=(E,{control program list address | (15)})

indicates the execute form of the macro instruction using a remote control program parameter list. Any problem program parameters specified are provided in a problem program parameter list expanded in line. The address of the control program parameter list can be coded as described under "address," or can be loaded into register 15, in which case SF=(E,(15)) should be coded.

16.8 MF=(E,{address | (1)}),SF=(E,{address | (15)})

indicates the execute form of the macro instruction using both a remote problem program parameter list and a remote control program parameter list. The addresses of the parameter lists are coded or loaded into registers 1 and 15, as explained above.

Note: If the STAI operand is specified in the execute form but the PURGE and ASYNCH suboperands are omitted, the control program assigns the default values QUIESCE and NO respectively. These default values replace any PURGE and ASYNCH values specified in the corresponding list form.

CALL -- Pass Control to a Control Section

17.1 The CALL macro instruction causes control to be passed to a control section at a specified entry point, as follows:

- OVERLAY: The overlay segment containing the designated entry point is brought into main storage if required, and control is passed to the segment. (15) must not be designated in an exclusive call. Refer to Linkage Editor and Loader, for details on overlay. The CALL macro instruction cannot be used in an asynchronous exit routine.

- NON-OVERLAY: If a symbol is designated, the load module containing that entry point will be included in the same load module as the CALL macro instruction by the linkage editor. When the CALL macro instruction is executed, control is passed to the control section at the specified entry point. If (15) is designated, the load module containing the entry point must be in main storage and register 15 must contain the address of the entry point.

17.2 The linkage relationship established when control is passed is the same as that created by a BAL instruction; that is, the issuing program expects control to be returned. The control program is not involved in passing control, so the reusability status of the called program must be maintained by the user.

17.3 An address parameter list can be constructed and a calling sequence identifier can be provided. The standard form of the CALL macro instruction is written as follows:

[symbol]	CALL	{entry point name / (15)} [,(address parameters)[,VL]] [,ID=number]

17.4 entry point name — Sym

is the name of the entry point to be given control; the name is used in the macro instruction as the operand of a V-type address constant. If (15) is designated, register 15 must contain the address of the entry point to be given control.

17.5 address parameters — A-type, (2-12)

are one or more address parameters, separated by commas, to be passed to the called program. Each address is expanded, in the order designated, to a fullword on a fullword boundary. When control is passed, register 1 contains the address of the first parameter. If no address parameters are designated, the contents of register 1 are not changed.

17.6 VL

is written as shown. It can be designated only if address parameters are designated. It should be used only when a variable number of parameters can be passed to the called program. VL causes the high-order bit of the last address parameter in the macro expansion to be set to 1; the bit can be checked by the called program to find the end of the list.

17.7 ID= — Sym, Dec Dig

maximum value is 2^{16}-1. The last fullword of the macro expansion is a NOP instruction containing the ID value in the low-order two bytes. Register 14 contains the address of the NOP instruction when the called program is given control.

CALL -- List Form

18.1 The list form of the CALL macro instruction is used to construct a problem program parameter list. This problem program parameter list can be referred to in the execute form of a CALL, LINK, ATTACH, or XCTL macro instruction.

18.2 The description of the standard form of the CALL macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are completely optional and which are required in at least one of a pair of list and execute forms. The format description below indicates the optional and required operands in the list form only. The comma before the parenthesis must be coded to indicate the absence of the entry point name operand, which is not allowed in the list form.

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

[symbol]	CALL	,(address parameters) [,VL],MF=L

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

MF=L
indicates the list form of the CALL macro instruction.

CALL -- Execute Form

19.1 A remote problem program parameter list is referred to and can be modified by the execute form of the CALL macro instruction.

19.2 The description of the standard form of the CALL macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the execute form only.

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

```
┌─────────┬─────────┬────────────────────────────────────────────────────────┐
│[symbol] │  CALL   │{entry point name}[,(address parameters) [,VL]]         │
│         │         │{       (15)     }                                      │
│         │         │                                                        │
│         │         │[,ID=number],MF=(E,{problem program list address})      │
│         │         │                   {  (1)                       }       │
└─────────┴─────────┴────────────────────────────────────────────────────────┘
```

name
: is any name valid in the assembler language.

address
: is any address that is valid in an RX-type instruction, or one of general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

number
: is any absolute expression that is valid in the assembler language, or one of general registers 2 through 12, previously loaded with the indicated value. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

MF=(E,{problem program list address})
      {   (1)   }
: indicates the execute form of the macro instruction using a remote problem program parameter list. If address parameters are also specified, they will be placed on contiguous fullword boundaries, beginning at the address specified in the MF operand and sequentially overlaying corresponding fullwords in the existing list. The address of the problem program parameter list can be coded as described under address, or can be loaded into register 1, in which case MF=(E,(1)) should be coded.

CHAP -- Change Dispatching Priority (MFT Without Subtasking)

20.1 When used in an operating system with MFT without subtasking, the CHAP macro instruction results in an effective NOP instruction. The CHAP macro instruction is used in an operating system with MFT without subtasking to assure compatability with an operating system with MVT or MFT with subtasking. The CHAP macro instruction is written as follows:

[symbol]	CHAP	priority change value [,tcb location address] [,'S']

CHAP

CHAP -- Change Dispatching Priority (MVT, MFT With Subtasking)

21.1 The CHAP macro instruction changes the dispatching priority of the task or any of its subtasks. The CHAP macro instruction may also change the limit priority of a subtask. (See Priority of Subtasks in Section I.)

21.2 Note: The limit priority of the job step task depends on the PRTY parameter of the JOB statement and the DPRTY parameter of the EXEC statement. The dispatching priority of any task determines whether or not the task participates in time slicing (only when the operating system includes the time-slicing option). For more details, refer to Section I.

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

[symbol]	CHAP	priority change value [,tcb location address] [,'S']

priority change value — Sym, Dec Dig, (0),(2-12)
is the signed value to be added to the dispatching priority of the specified task. If the value is negative and contained in a register, it should be in two's complement form.

tcb location address — RX-type, (1-12)
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 instead of an address, it indicates that the priority of the active task is to be changed. 'S' is assumed if the operand is omitted or if it is coded to specify a zero address.

CHKPT -- Take Checkpoint for Restart Within a Job Step

22.1 The CHKPT macro instrucion establishes a checkpoint for the job step. If the step terminates abnormally, it is automatically restarted from the checkpoint. On restart, execution resumes with the instruction that follows the CHKPT macro instruction. If the step again terminates abnormally (before taking another checkpoint), it is again restarted from the checkpoint. When several checkpoints are taken, the step is automatically restarted from the most recent checkpoint.

22.2 Automatic restart from a checkpoint is suppressed if:

1. The job step completion code is not one of a set of codes specified at system generation.
2. The operator does not authorize the restart.
3. The restart definition parameter of the JOB or EXEC statement specifies no restart (RD=NR) or no checkpoint (RD=NC or RD=RNC).
4. The CANCEL operand appears in the last CHKPT macro instruction issued before abnormal termination.

Under any of these conditions, automatic checkpoint restart does not occur. Automatic step restart (restart from the beginning of the job step) can occur, except under condition 1 or 2, or when the job step was restarted from a checkpoint prior to abnormal termination. Automatic step restart is requested through the restart definition parameter of the JOB or EXEC statement (RD=R or RD=RNC).

22.3 When automatic restart is suppressed or unsuccessful, a deferred restart can be requested by submitting a new job. The new job can specify restart from the beginning of the job step or from any checkpoint for which there is an entry in the checkpoint data set.

22.4 The checkpoint data set contains the information necessary to restart the job step from a checkpoint. The control program records this information when the CHKPT macro instruction is issued. The macro instruction refers to the data control block for the data set, which must be on a magnetic tape or direct access volume. A tape can have standard labels, nonstandard labels, or no labels.

22.5 If the checkpoint data set is not open when the CHKPT macro instruction is issued, the control program opens the data set and then closes it after writing the checkpoint entry. If the data set is physically sequential and is opened by the control program, the checkpoint entry is written over the previous entry in the data set, unless the DD statement specifies DISP=MOD. By writing entries alternately into two checkpoint data sets, it is possible to keep the entries for the two most recent checkpoints while deleting those for earlier checkpoints.

22.6 The data control block for the checkpoint data set must specify:

DSORG=PS or PO, RECFM=U or UT, MACRF=(W), BLKSIZE=nnn, and
DDNAME=any name

where nnn is at least 600 bytes, but not more than 32,760 bytes for magnetic tape, and not more than the track length for direct access. (If the data set is opened by the control program, block size need not be specified; the device-determined maximum block size is assumed if no block size is specified.) For seven-track tape, the data control block must specify TRTCH=C; for direct access, it must specify or imply

KEYLEN=0. To request chained scheduling, OPTCD=C and NCP=2 must be specified. With direct access, OPTCD=W can be specified to request validity checking for write operations, and OPTCD=WC can be specified to combine validity checking and chained scheduling.

22.7 The standard form of the CHKPT macro instruction is written as shown below:

```
[symbol] | CHKPT | {dcb address} [,checkid address] [,checkid length]
         |       | {CANCEL     }                    [,'S'           ]
```

22.8 dcb address — A-type, (2-12)

is the address of the data control block for the checkpoint data set.

22.9 checkid address — A-type, (2-12)

is the address of the checkpoint identification field. The contents of the field are used when the job step is to be restarted from the checkpoint. They are used by the control program in requesting operator authorization for automatic restart, and by the programmer in requesting deferred restart.

22.10 If the next operand specifies the length of the field (checkid length), or if it is omitted to imply a length of eight bytes, the field must contain the checkpoint identification when the CHKPT macro instruction is issued. If the next operand is written as 'S', the identification is generated and placed in the field by the control program. If both operands are omitted, the control program generates the identification, but does not make it available to the problem program. In each case, the identification is written in a message to the operator.

22.11 The control program writes the checkpoint identification as part of the entry in the checkpoint data set. For a sequential data set, the identification can be any combination of up to 16 letters, digits, printable special characters, and blanks. For a partitioned data set, it must be a valid member name of up to eight letters and digits, starting with a letter. The identification for each checkpoint should be unique.

22.12 If the control program generates the identification, the identification is eight bytes in length. It consists of the letter C followed by a seven-digit decimal number. The number is the total number of checkpoints taken by the job, including the current checkpoint, checkpoints taken earlier in the job step, and checkpoints taken by any previous job steps.

22.13 checkid length — Sym, Dec Dig, (2-12)

is the length in bytes of the checkpoint identification field. The maximum length is 16 bytes when the checkpoint data set is physically sequential, 8 bytes when it is partitioned. For a partitioned data set, the field can be longer than the actual identification, if the unused portion is blank. If the operand is omitted, the implied length is eight bytes.

22.14 The control program supplies the checkpoint identification if 'S' is coded instead of a field length. The implied field length is eight bytes.

22.15 CANCEL
cancels the request for automatic restart from the most recent checkpoint. If another checkpoint is taken before abnormal termination, the job step can be restarted at that checkpoint.

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

Hex Code	Meaning
00	Successful completion. A valid checkpoint entry was written, or checkpoint was suppressed in the JOB or EXEC statement (RD=NC or RD=RNC).
04	Successful restart. The macro instruction was used earlier to take a checkpoint, and the job step has now been restarted from that checkpoint. If the job step terminates abnormally before establishing another checkpoint, the job step may again be restarted (automatically) from the same checkpoint.
08	Unsuccessful completion. No checkpoint entry was written due to one of the following conditions: • The parameters passed by the CHKPT macro instruction were invalid. • The CHKPT macro instruction was issued by an exit routine (other than a end-of-volume exit routine). • A STIMER macro instruction has been issued, and the time interval has not been completed. • A WTOR macro instruction has been issued, and the reply has not been received. • The checkpoint data set is on a direct access volume and is full. Secondary space was allocated but not used. (Secondary space cannot be used for a checkpoint data set. However, had it not been requested, the job step would have been abnormally terminated.) • In a system with MVT or MFT with subtasking, the job step comprises more than one task. • In a system with MVT, the job step has been allocated storage through the rollout/rollin option. • A graphics-type DSORG has been found in an open DCB. Graphic devices are not supported in Checkpoint/Restart.
0C	Unsuccessful completion. The checkpoint entry is invalid due to an uncorrectable output error, or no entry was written due to one of the following conditions: • There was no DD statement for the checkpoint data set. • There was an uncorrectable error in completing input/output operations that were begun before the CHKPT macro instruction was issued.

Hex Code	Meaning (Cont'd)
10	Successful completion with possible error condition. A valid checkpoint entry was written, but the task has control of a serially reusable resource. The task will not have control of this resource if the job step is restarted from the checkpoint.
14	Unsuccessful completion. An end-of-volume was detected at a critical point in writing the checkpoint data set to tape. A volume switch did occur, however, and the CHKPT macro instruction may be reissued immediately to get the entry written on the new volume. If a programmer-specified checkid was used, it is advisable to use a new checkid when reissuing CHKPT.

CHKPT -- List Form

23.1 The list form of the CHKPT macro instruction is used to construct a control program parameter list.

23.2 The description of the standard form of the CHKPT macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the list form only. Note that the CANCEL operand, which can be coded in the standard form, cannot be coded in the list form.

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

[symbol]	CHKPT	[dcb address],[checkid address], checkid length ,MF=L 'S'

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

length
is any absolute expression that is valid in the assembler language.

MF=L
indicates the list form of the CHKPT macro instruction.

CHKPT -- Execute Form

24.1 A remote control program parameter list is referred to, and can be modified by, the execute form of the CHKPT macro instruction.

24.2 The description of the standard form of the CHKPT macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands for the execute form only. Note that the CANCEL operand, which can be coded in the standard form, cannot be coded in the execute form.

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

```
┌──────────┬─────────┬──────────────────────────────────────────────────────────┐
│[symbol]  │ CHKPT   │ [dcb address],[checkid address],┌checkid length┐          │
│          │         │                                 └'S'           ┘          │
│          │         │ ,MF=(E,{control program list address})                   │
│          │         │        {    (1)                      }                   │
└──────────┴─────────┴──────────────────────────────────────────────────────────┘
```

address
: is any address that is valid in an RX-type instruction, or one of the general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

length
: is any absolute expression that is valid in the assembler language, or one of the general registers 2 through 12, previously loaded with the indicated value. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

```
MF=(E,{control program list address})
      {    (1)                      }
```

indicates the execute form of the macro instruction using a remote control program parameter list. The address of the control program parameter list can be coded as described under address, or can be loaded into register 1, in which case MF=(E,(1)) should be coded.

DELETE

DELETE -- Relinquish Control of a Load Module

25.1 The DELETE macro instruction cancels the effect of a single previous LOAD request for the designated load module by reducing the count of outstanding LOAD requests by one. If this DELETE macro instruction cancels the only outstanding LOAD request for the module, and no other requirements exist for the module, the main storage area occupied by the load module is made available for reassignment by the control program. The name specified in the DELETE macro instruction must be the same name as that specified in the LOAD macro instruction, which was issued to bring the load module into main storage; it must be issued in performance of the same task as the LOAD macro instruction.

25.2 The DELETE macro instruction is written as follows:

symbol	DELETE	{ EP=symbol EPLOC=address of name DE=address of list entry }

EP= Sym
: is the entry point name that was used to bring the module into main storage.

EPLOC= RX-type, (0,2-12)
: is the address of the entry point name described above. The name must be padded with blanks to eight bytes, if necessary.

DE= RX-type, (0,2-12)
: is the address of the name field of a list entry for the entry point name described above, constructed using the BLDL macro instruction.

25.3 When control is returned, register 15 contains a 0 if the operation was completed successfully. Register 15 contains a 4 if a LOAD macro instruction was not issued for the task issuing the DELETE macro instruction or if the number of outstanding LOAD requests had previously become zero.

DEQ -- Release a Serially Reusable Resource

26.1 The DEQ macro instruction is used to remove control of one or more (maximum is 255) serially reusable resources from the active task. It can also be used to determine whether control of the resource is currently assigned to or requested for the active task. Register 15 is set to zero if the request is satisfied.

26.2 In a system with MVT, the DEQ macro instruction must be used to release control of every resource assigned through the use of the ENQ macro instruction; if normal termination of the task is attempted while the task still has control of any of the resources assigned through an ENQ macro instruction, the task is abnormally terminated. In a system with either MFT or MVT, an unconditional request to remove control of a resource from a task that is not in control of the resource results in abnormal termination of the task.

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

```
[symbol] | DEQ | (qname address,rname address,[rname length],⎡STEP  ⎤,...)
         |     |                                            ⎣SYSTEM⎦
         |     |
         |     | [,RET=HAVE]
```

26.4 qname address — A-type, (2-12)

is the address in main storage of an eight-character name. Every program issuing a request for a serially reusable resource must use the same qname and rname to represent the resource. The name should not start with SYS, so that it will not conflict with system names. The name must be the same qname previously specified for the resource in an ENQ macro instruction.

26.5 rname address — A-type, (2-12)

is the address in main storage of the name used in conjunction with the qname to represent the resource in a previous ENQ macro instruction. The name can be qualified and must be from 1 to 255 bytes long.

26.6 rname length — Sym, Dec Dig, (2-12)

is the length of the rname described above. The length must have the same value as specified in the previous ENQ macro instruction. If the operand is omitted, the assembled length of the rname is used. A value between 1 and 255 can be specified to override the assembled length, or a value of zero can be specified. If zero is specified, the length of the rname must be contained in the first byte at the rname address designated above.

26.7 STEP or SYSTEM

is written as shown. The same STEP or SYSTEM option must be designated as was selected in the ENQ macro instruction issued for the resource.

26.8 RET=HAVE

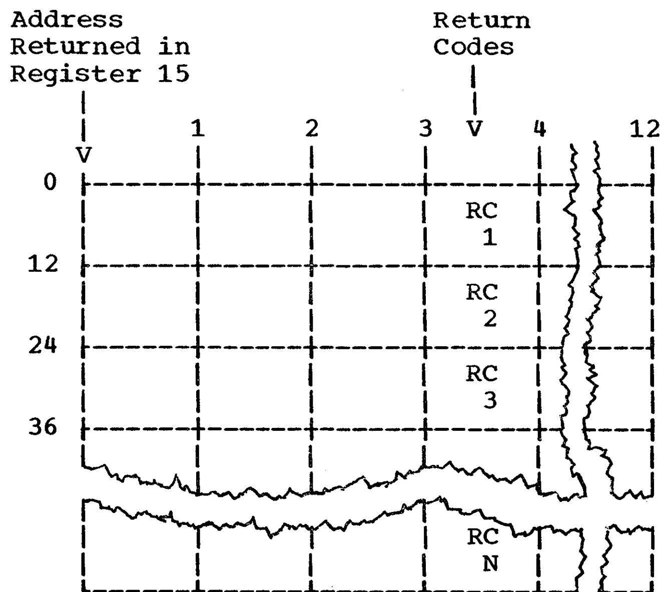
is written as shown. It specifies that the request for release of control of all the resources named in the DEQ macro instruction is to be honored only if the active task has been assigned control of the resources. If the operand is omitted, the request for release is unconditional, and the active task is abnormally terminated if it has not been assigned control of the resources. The results of conditional requests are indicated by the return codes shown in Figure 59.

DEQ

26.9 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 the DEQ macro instruction are zero, register 15 contains zero. If any of the return codes are not zero, register 15 contains the address of a main storage area containing the return codes as shown in Figure 60. 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 in Figure 59.

Code	Meaning
0	Control of the resource has been released.
4	Control of 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 the DEQ macro instruction 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 control has previously been returned.

Figure 59. DEQ macro instruction return codes



Return codes are 12 bytes apart, starting 3 bytes from the address in register 15

Figure 60. Location of return codes in main storage

DEQ -- List Form

27.1 The list form of the DEQ macro instruction is used to construct a control program parameter list. Up to 255 resources can be specified in the DEQ macro instruction; therefore, the number of qname and rname combinations in the list form of the DEQ macro instruction must be equal to the maximum number of qname and rname combinations in any execute form of the macro instruction.

27.2 The description of the standard form of the DEQ macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the list form only.

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

```
┌─────────┬────────┬──────────────────────────────────────────────────────┐
│[symbol] │  DEQ   │([qname address],[rname address],[rname length],      │
│         │        │┌SYSTEM┐,...)     [,RET=HAVE],MF=L                    │
│         │        │└STEP  ┘                                              │
└─────────┴────────┴──────────────────────────────────────────────────────┘
```

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

length
  is any absolute expression valid in the assembler language.

MF=L
  indicates the list form of the DEQ macro instruction.

DEQ - E Form

DEQ -- Execute Form

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

28.2 The description of the standard form of the DEQ macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which operands are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the execute form only.

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

```
┌──────────┬─────────┬──────────────────────────────────────────────────────┐
│ [symbol] │  DEQ    │[([qname address],[rname address],[rname length],     │
│          │         │[SYSTEM],...)]    [,RET=HAVE]                         │
│          │         │[STEP  ]          [,RET=NONE]                         │
│          │         │                                                      │
│          │         │,MF=(E,{control program list address})                │
│          │         │       {  (1)                        }                │
└──────────┴─────────┴──────────────────────────────────────────────────────┘
```

28.4 address
: is any address that is valid in an RX-type instruction, or one of general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

28.5 length
: is any absolute expression that is valid in the assembler language, or one of general registers 2 through 12, previously loaded with the indicated value. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

28.6 RET=NONE
: specifies an unconditional request to release control of all of the resources. The request is processed as though no RET operand had been coded.

28.7 MF=(E,{control program list address}) / {(1)}
: indicates the execute form of the macro instruction using a remote control program parameter list. The address of the control program parameter list can be coded as described under address, or can be loaded into register 1, in which case MF=(E,(1)) should be coded.

DETACH -- Delete a Subtask (MFT Without Subtasking)

29.1 When used in an operating system with MFT without subtasking, the DETACH macro instruction results in an effective NOP instruction. This assures compatibility with an operating system with MVT or MFT with subtasking. The DETACH macro instruction is written as follows:

[symbol]	DETACH	tcb location address[,STAE={YES | NO}]

DETACH

DETACH -- Delete a Subtask (MFT With Subtasking)

30.1 The DETACH macro instruction is used to remove from the system a subtask created by an ATTACH macro instruction that specified the ECB or ETXR operand. Each subtask created in this manner must be removed from the system before the originating task terminates. Failure to remove these subtasks causes abnormal termination of originating task and all of its subtasks. Issuing a DETACH that specifies a subtask created without the ECB or ETXR operand also causes abnormal termination of the originating task when the specified subtask has already terminated. Issuing a DETACH that specifies a subtask that has not terminated causes termination of that subtask and all of its subtasks. A DETACH macro instruction can be issued only for subtasks created by the active task.

30.2 The DETACH macro instruction is written as follows. The operand in the shaded area is used only in an operating system with MVT. It is ignored if coded in an operating system with MFT:

[symbol]	DETACH	tcb location address [,STAE={YES | NO}]

tcb location address RX-type, (1-12)
is the main storage address of a fullword on a fullword boundary. The fullword contains the address of the task control block for the subtask to be removed from the system.

DETACH -- Delete a Subtask (MVT)

31.1 The DETACH macro instruction is used to remove from the system a subtask created by an ATTACH macro instruction that specified the ECB or ETXR operand. Each subtask created in this manner must be removed from the system before the originating task terminates. Failure to remove these subtasks causes abnormal termination of originating task and all of its subtasks. Issuing a DETACH that specifies a subtask created without the ECB or ETXR operand also causes abnormal termination of the originating task when the specified subtask has already terminated. Issuing a DETACH that specifies a subtask that has not terminated causes termination of that subtask and all of its subtasks. A DETACH macro instruction can be issued only for subtasks created by the active task.

31.2 The DETACH macro instruction is written as follows:

[symbol]	DETACH	tcb location address [,STAE={YES | NO}]

tcb location address RX-type, (1-12)
is the main storage address of a fullword on a fullword boundary. The fullword contains the address of the task control block for the subtask to be removed from the system.

STAE=

YES
indicates that the exit routine specified in a STAE macro instruction issued by the subtask is to be given control if the subtask is scheduled for abnormal termination while it is being detached. If a retry routine is specified by the STAE exit routine, it will not be given control.

NO
indicates that the exit routine specified in the STAE macro instruction will not be given control if the subtask is scheduled for abnormal termination (ABEND) while it is being detached. If neither YES nor NO is specified, NO is assumed.

DOM -- Delete Operator Message (Without the Multiple Console Support (MCS) Option)

32.1 When used in an operating system without the Multiple Console Support (MCS) option, the DOM macro instruction results in an effective NOP instruction. This assures compatibility with an operating system that has the MCS option. The DOM macro instruction is written as follows:

[symbol]	DOM	{MSG=register {MSGLIST=address}

DOM -- Delete Operator Message (With the Multiple Console Support (MCS) Option)

33.1 The DOM macro instruction causes the deletion of a message or group of messages from a display operator console. When a program no longer requires that a message be displayed, a DOM macro instruction should be issued to delete the message.

33.2 When a WTO or WTOR macro instruction is executed, the operating system assigns an identification number to the message. The operating system returns the assigned identification number (24 bits and right-justified) to the issuing program in general register 1. When display of the message is no longer needed, the DOM macro instruction should be coded using the identification number that was returned in general register 1.

33.3 The DOM macro instruction is written as follows:

[symbol]	DOM	{MSG=register {MSGLIST=address}

MSG=
: specifies a general register from 1 through 12 that contains the 24-bit, right-justified identification number of the message to be deleted. This operand is used for the deletion of a single message. If register 1 is used, the macro expansion is shortened by two bytes.

MSGLIST=
: specifies the address of a list of one or more fullwords, each word containing a 24-bit, right-justified identification number of a message to be deleted. A maximum of 60 identification numbers may be contained in the message list. If more than 60 identification numbers are contained in the list, the list will be truncated after the 60th number. The list must begin on a fullword boundary. The end of the list must be indicated by setting the high order bit of the last fullword entry to 1. If register 1 is used, the macro expansion is shortened by four bytes. If any register 2 through 12 is used, the macro expansion is shortened by two bytes.

DXR -- Divide Extended Register

34.1 The DXR macro instruction is used to divide one extended-precision floating-point number by another. A detailed description of the division process is given in the section on Extended Precision and Rounding in Principles of Operation.

34.2 To use the DXR macro instruction, the user must have provided a SPIE Exit routine to process the program exception caused (intentionally) by the execution of the DXR macro instruction. The SPIE Exit routine is described in Section I under Extended-Precision Floating-Point Simulation.

34.3 The DXR macro instruction is written as follows:

[symbol]	DXR	reg1,reg2

reg1 — Sym, Dec Dig
is the register that contains the dividend. The quotient is placed in this register; the remainder is discarded.

reg2 — Sym, Dec Dig
is the register that contains the divisor.

Notes: Following is a list of limitations that apply to both the reg1 and reg2 operands:

- Registers 0 and 4 are the only registers that can be specified. However, the registers can be specified in either order.
- The registers must be specified as decimal digits 0 or 4 or as symbols that have been equated to these decimal digits.
- The registers are never coded within parentheses.

ENQ -- Request Control of a Serially Reusable Resource

35.1 The ENQ macro instruction requests the control program to symbolically assign control of one or more serially reusable resources to the active task. Each resource is represented by a unique qname/rname combination. The control program does not correlate the qname/rname combination with an actual resource. Thus, access to a resource is logically, not physically restricted. That is, tasks may use a serially reusable resource without using the ENQ macro instruction, but in doing so may jeopardize program reliability.

35.2 If any of the resources are not available (that is, have been specified in an exclusive ENQ request and not specified in a subsequent DEQ request) and this is an unconditional request, the active task is placed in a wait condition until all of the requested resources are available. If the ENQ request is conditional, control is immediately returned to the active task. Once control of a resource is symbolically assigned to a task, it remains with that task until one of the programs of that task issues a DEQ macro instruction specifying the same resource.

35.3 The ENQ macro instruction may also be used to determine the status of a resource; that is, whether the resource is immediately available or in use, and whether control has been previously requested for the active task in another ENQ macro instruction.

35.4 Issuing an unconditional request for a resource currently allocated to the active task (by a previous ENQ without an intervening DEQ) results in abnormal termination of the task. If normal termination of a task is attempted while the task still has control of any serially reusable resources, the task is abnormally terminated.

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

[symbol]	ENQ	(qname address, rname address,[E / S],[rname length], [SYSTEM / STEP],...) [,RET=TEST / ,RET=USE / ,RET=HAVE / ,RET=CHNG]

35.6 qname address — A-type, (2-12)

is the address in main storage of an eight-character name. Every program issuing a request for a serially reusable resource must use the same qname and rname to represent the resource. The name should not start with SYS, so that it will not conflict with system names.

35.7 rname address — A-type, (2-12)

is the address in main storage of the name used in conjunction with the qname to represent the resource. The name can be qualified and must be from 1 to 255 bytes long.

Note: Because the control program does not correlate the qname/rname combination with the resource, protection against concurrent use of a serially reusable resource is not provided unless all tasks use the ENQ function.

ENQ

35.8 E

is written as shown. It specifies that the request is for exclusive control of the resource. If the operand is omitted, a request for exclusive control is assumed. If the resource is modified while under control of the task, the request must be for exclusive control.

35.9 S

is written as shown. It specifies that the request is for shared control of the resource. If the resource is not modified while under control of the task, the request should be for shared control.

35.10 rname length — Sym, Dec Dig (2-12)

is the length of the rname described above. If the operand is omitted, the assembled length of the rname is used. A value between 1 and 255 can be specified to override the assembled length, or a value of zero can be specified. If zero is specified, the length of the rname must be contained in the first byte at the rname address designated above.

35.11 STEP

is written as shown. It specifies that the resource is used only within the job step of the issuing program, and that a request for the same qname and rname from a program in another job step denotes a different resource. This option is assumed if the operand is omitted.

35.12 SYSTEM

is written as shown. It specifies that the resource may be used by programs of more than one job step, and that requests for the same qname and rname from programs of other job steps in the system denote the same resource.

Because SYSTEM and STEP are opposite in meaning, both cannot refer to the same resource. If two macro instructions specify the same qname and rname, but one specifies SYSTEM and the other specifies STEP, they are treated as requests for different resources. Conversely, when one resource is used by a single job step and another is used by several job steps, the same qname and rname can be used for both.

35.13 RET=

specifies a conditional request for all of the resources named in the ENQ macro instruction. If the operand is omitted, the request is unconditional. The results of a conditional request are indicated by the return codes described in Figure 61; the types of conditional requests are as follows:

TEST - tests the availability status of the resources but does not request control of the resources.

USE - specifies that control of the resources 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.

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

CHNG - requests a change of the attribute from shared to exclusive for a resource which is controlled by the active task.

35.14 Return codes are provided by the control program only if RET=TEST, RET=USE, RET=HAVE, or RET=CHNG, is designated; 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 zero, register 15 contains zero. If any of the return codes are not zero, register 15 contains the address of a main storage area containing the return codes, as shown in Figure 62. 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 in Figure 61.

Code	Meaning			
	RET=TEST	RET=USE	RET=HAVE	RET=CHNG
0	The resource is immediately available.	Control of the resource has been assigned to the active task.		Control of the resource is now assigned exclusively to the active task.
4	The resource is not immediately available.		---	Requested attribute change cannot be made at this time.
8	A previous request for control of the same resource has been made for the same task			The active task is not in control of the resource.

Figure 61. ENQ macro instruction return codes

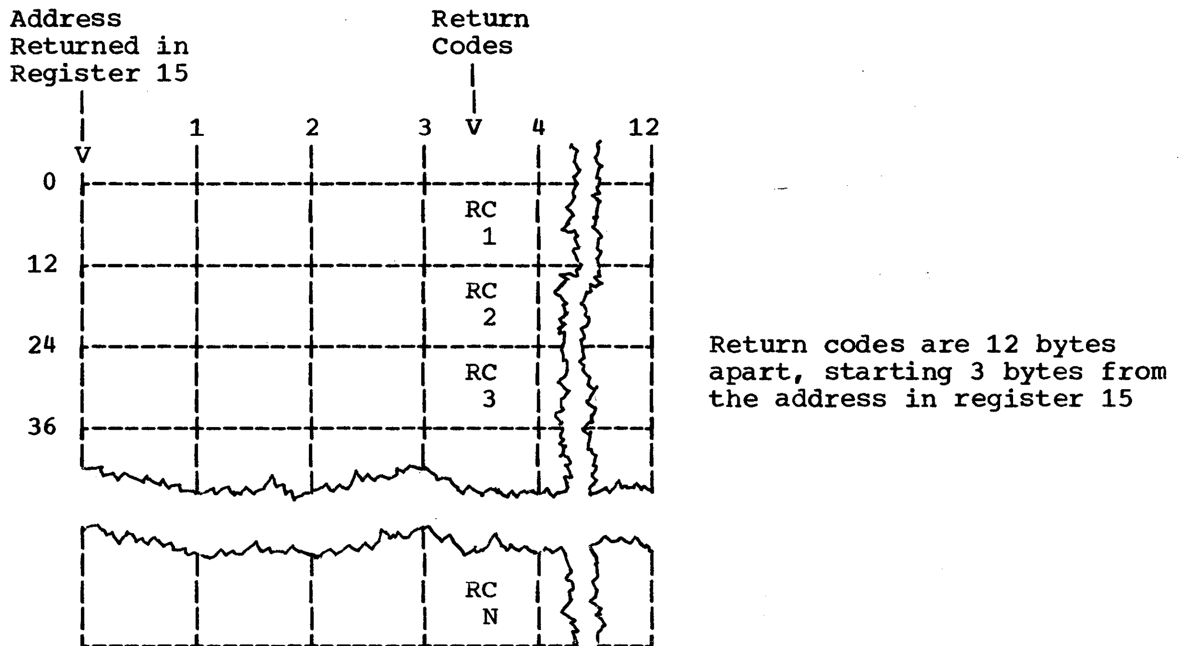


Figure 62. Location of return codes in main storage

ENQ - L Form

ENQ -- List Form

36.1 The list form of the ENQ macro instruction is used to construct a control program parameter list. Any number of resources can be specified in the ENQ macro instruction; therefore, the number of qname and rname combinations in the list form of the ENQ macro instruction must be equal to the maximum number of qname and rname combinations in any execute form of the macro instruction.

36.2 The description of the standard form of the ENQ macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the list form only.

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

```
[symbol]   ENQ   ([qname address],[rname address],[E],[rname length],
                                                  [S]
                 [SYSTEM],...)   [,RET=HAVE],MF=L
                 [STEP  ]        [,RET=TEST]
                                 [,RET=USE ]
                                 [,RET=CHNG]
```

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

length
    is any absolute expression valid in the assembler language.

MF=L
    indicates the list form of the ENQ macro instruction.

ENQ -- Execute Form

37.1 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 the ENQ macro instruction.

37.2 The description of the standard form of the ENQ macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which operands are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the execute form only.

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

```
[symbol] | ENQ | [([qname address],[rname address],[E],[rname length],
         |     |                                   [S]
         |     | [SYSTEM],...)]   [,RET=HAVE]
         |     | [STEP  ]         [,RET=TEST]
         |     |                  [,RET=USE ]
         |     |                  [,RET=NONE]
         |     |                  [,RET=CHNG]
         |     |
         |     | ,MF=(E,{control program list address})
         |     |        {   (1)                      }
```

address
: is any address that is valid in an RX-type instruction, or one of general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

length
: is any aboslute expression that is valid in the assembler language, or one of general registers 2 through 12, previously loaded with the indicated value. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

RET=NONE
: specifies an unconditional request for control of all of the resources. The request is processed as though no RET operand had been coded in the list form.

MF=(E,{control program list address}
      {   (1)   })
: indicates the execute form of the macro instruction using a remote control program parameter list. The address of the control program parameter list can be coded as described under "address," or can be loaded into register 1, in which case MF=(E,(1)) should be coded.

EXTRACT -- Provide Information From TCB Fields (MFT Without Subtasking)

38.1 The EXTRACT macro instruction causes the control program to provide the address of the task input-output table for the job step, or the address of the command scheduler communications list, or both.

38.2 The standard form of the EXTRACT macro instruction is written as shown in the format description below. The operands in the shaded area of the format description are used only in an operating system with MVT or MFT with subtasking; they are ignored if coded in an operating system with MFT without subtasking. The operands in the nonshaded area can be coded with any configuration of the operating system, although some of the fields available do not apply to MFT without subtasking.

```
[symbol] | EXTRACT | answer area address [,tcb location address],
         |         |                     [,'S'                 ]
         |         | FIELDS=(codes)
```

answer area address — A-type, (2-12)
is the main storage address of one or more consecutive fullwords, starting on a fullword boundary. One fullword is required for each parameter coded in the FIELDS operand, unless FIELD=(ALL) is coded. FIELDS=(ALL) requires seven fullwords.

FIELDS=
TIOT or ALL can be coded to obtain the address of the task input-output table. If TIOT is coded, the address is returned by the control program, right-adjusted, in the fullword answer area. If ALL is coded, the address is placed, right-adjusted, in the seventh fullword.

COMM can also be coded to obtain the address of the command scheduler communications area. If COMM is coded with TIOT, the address is returned in the second fullword. If COMM is coded with ALL, the address is returned in the eighth fullword. If COMM is coded without TIOT or ALL, the address is returned in the first fullword.

38.3 Note: Additional fields can be obtained when the EXTRACT macro instruction is used in an operating system with MVT or MFT with subtasking. If an EXTRACT macro instruction requesting these additional fields is used in an operating system with MFT without subtasking, an additional fullword is required in the answer area for each field. The control program sets each of the additional fullwords to zero.

EXTRACT -- Provide Information From TCB Fields (MVT, MFT With Subtasking)

39.1 The EXTRACT macro instruction causes the control program to provide information from specified fields of the task control block or a subsidiary control block for either the active task or one of its subtasks. The information is placed in an area provided by the problem program in the order shown in Figure 63. The standard form of the EXTRACT macro instruction is written as follows:

```
┌─────────┬────────┬──────────────────────────────────────────────────────┐
│[symbol] │EXTRACT │answer area address ┌,tcb location address┐,         │
│         │        │                    └,'S'                 ┘          │
│         │        │FIELDS=(codes)                                        │
└─────────┴────────┴──────────────────────────────────────────────────────┘
```

39.2 answer area address — A-type, (2-12)

is the address in main storage of one or more consecutive fullwords, starting on a fullword boundary. The number of fullwords required is the same as the number of fields specified in the FIELDS operand, unless FIELDS=(ALL) is coded. FIELDS=(ALL) requires seven fullwords.

39.3 tcb location address — A-type, (2-12)

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 instead of an address, it indicates that information is requested from the task control block for the active task. 'S' is assumed if the operand is omitted or if it is coded to specify a zero address.

39.4 FIELDS=

is one or more of the following sets of characters, written in any order and separated by commas, which are used to request the associated task control block information. The information from the requested field is returned in the relative order shown in Figure 63; if the information from a field is not requested, the associated fullword is omitted. If ALL is specified, the answer area will include all the fields in Figure 63 from GRS to TIOT, including the reserved word. Addresses are always returned in the low-order three bytes of the fullword, and the high-order byte is set to zero. Fields for which no address or value has been specified in the task control block are set to zero.

ALL
: requests information from the GRS, FRS, RESERVED, AETX, PRI, CMC, and TIOT fields.

GRS
: the address of the general register save area used by the control program to save the general registers (in the order of 0 through 15) when the task is not active.

FRS
: the address of the floating point register save area used by the control program to save the floating point registers (in the order of 0, 2, 4, 6) when the task is not active.

AETX
: the address of the end of task exit routine specified in the ATTACH macro instruction used to create the task.

EXTRACT

PRI
the current limit (third byte) and dispatching (fourth byte) priorities of the task. The two high-order bytes are set to zero.

CMC
the task completion code. If the task is not complete, the field is set to zero.

TIOT
the address of the task input/output table.

COMM
the address of the command scheduler communications list. The list consists of a pointer to the communications event control block and a pointer to the command input buffer. The high-order bit of the last pointer is set to one to indicate the end of the list.

PSB
the address of the protected storage control block (PSCB), which is extracted from the job step control block (JSCB). This field is meaningful only for jobs running in a time sharing environment.

TSO
the address of the time sharing flags field in the task control block (TCB). This field is meaningful only for jobs running in a time sharing environment.

TJID
the terminal job identifier (TJID) of the task specified in the tcb location address operand. This field is meaningful only for jobs running in a time sharing environment.

39.5

Note: The user must provide an answer area consisting of contiguous fullwords, one for each of the codes specified in the FIELDS operand, with the exception of the ALL code. If ALL is specified, the user must provide a 7-word answer area to accommodate the GRS, FRS, RESERVED, AETX, PRI, CMC and TIOT fields. The ALL code does not include the COMM, PSB, TSO, and TJID codes.

For example, if FIELDS=(TIOT,GRS,PRI,TSO,PSB,TJID) is coded, a 6-fullword answer area address is required, and the extracted information will appear in the answer area in the same relative order as shown in Figure 63. (That is, GRS will be returned in the first word. PRI in the second word, TIOT in the third word, etc.)

If FIELDS=(ALL,TSO,PSB,COMM,TJID) is coded, an 11-word answer area is required, and the extracted information will appear in the answer area in the relative order shown in Figure 63.

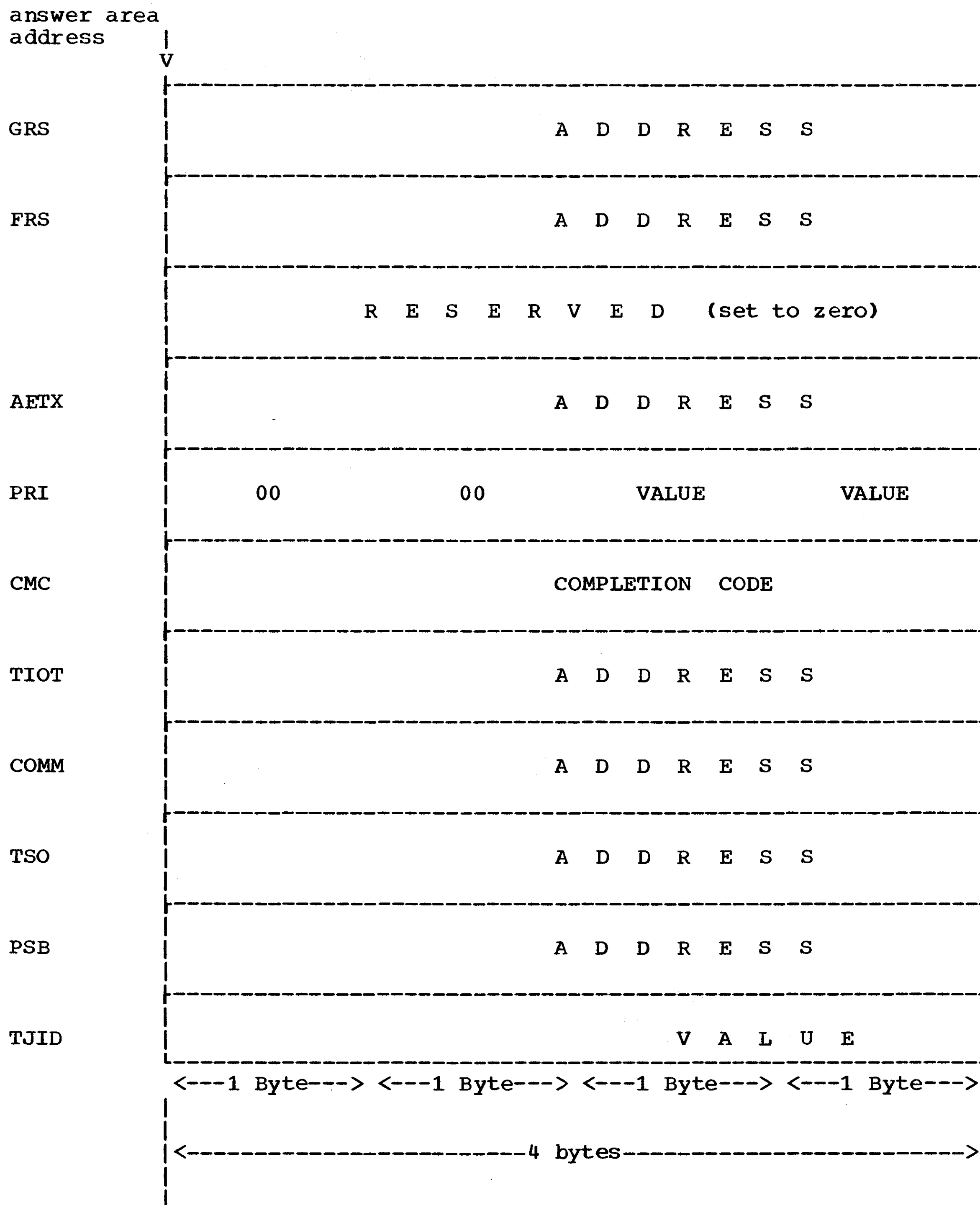


Figure 63. EXTRACT answer area field order

EXTRACT - L Form

EXTRACT -- List Form

40.1 The list form of the EXTRACT macro instruction is used to construct a control program parameter list.

40.2 The description of the standard form of the EXTRACT macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the list form only.

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

```
┌─────────┬────────┬─────────────────────────────────────────────┐
│[symbol] │EXTRACT │[answer area address][,{tcb location address}] │
│         │        │                       {'S'                 } │
│         │        │[,FIELDS=(codes)],MF=L                       │
└─────────┴────────┴─────────────────────────────────────────────┘
```

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

codes
    are one or more of the sets of characters defined in the description of the standard form of the macro instruction. Each use of the FIELDS operand in the execute form overrides any previous codes.

MF=L
    indicates the list form of the EXTRACT macro instruction.

EXTRACT -- Execute Form

41.1 A remote control program parameter list is referred to, and can be modified by, the execute form of the EXTRACT macro instruction.

41.2 The description of the standard form of the EXTRACT macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the execute form only.

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

```
[symbol] EXTRACT [answer area address][{,tcb location address}]
                                       { 'S'                  }
                 [,FIELDS=(codes)]
                 ,MF=(E,{control program list address})
                        { (1)                         }
```

address
    is any address that is valid in an RX-type instruction, or one of general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

codes
    are one or more of the sets of characters defined in the description of the standard form of the macro instruction. If the FIELDS operand is used in the execute form, any codes specified in a previous FIELDS operand are cancelled and must be respecified if required for this execution of the macro instruction.

MF=(E,{control program list address})
      {(1)   }
    indicates the execute form of the macro instruction using a remote control program parameter list. The address of the control program parameter list can be coded as described under address, or can be loaded into register 1, in which case MF=(E,(1)) should be coded.

FREEMAIN

FREEMAIN -- Release Allocated Main Storage (MFT)

42.1 The FREEMAIN macro instruction releases one area of main storage that had previously been allocated to the job step or subtask as a result of a GETMAIN macro instruction. The main storage area must start on a doubleword boundary. The requesting task is abnormally terminated if the specified main storage area does not start on a doubleword boundary, if the main storage area is not currently allocated to the requesting task, or if a request is made to release zero bytes.

42.2 The control program does not use the main storage area at the address in register 13 as a save area when processing release requests if R is coded.

42.3 The standard form of the FREEMAIN macro instruction is written as shown in the format description below. The operands in the shaded area of the format description are used only in an operating system with MVT. If they are coded in an operating system with MFT, and are coded correctly, they are ignored; if they are not coded correctly, the job step is abnormally terminated. The operands in the nonshaded area can be coded with any configuration of the operating system.

[symbol]	FREEMAIN	E,LV=number,A=address[,SP=number] R,LV=(0),A=address R,LV=(0),A=(1) R,LV=number,A=address[,SP=number] R,LV=number,A=(1)[,SP=number] V,A=address[,SP=number]
Note: only those operand combinations indicated above are valid		

42.4 E

(element) written as shown; specifies release of a single area of allocated main storage, with a length indicated by the LV operand. The address of the allocated main storage area is provided at the address indicated in the A operand.

42.5 R

(register) written as shown; specifies release of a single area of allocated main storage, with a length indicated by the LV operand. The address of the allocated main storage area is provided by the A operand.

42.6 V

(variable) written as shown; specifies release of a single area of allocated main storage. The address and length of the main storage area are provided at the address indicated in the A operand.

42.7 LV= Sym, Dec Dig, (2-12)

is the length, in bytes, of the main storage area being released. The value should be a multiple of eight; if it is not, the control program uses the next higher multiple of eight. If R is designated, LV=(0) may be designated; the low-order three bytes of register 0 must contain the length. The contents of the high-order byte are ignored. (With MVT, the high-order byte contains the subpool number.)

42.8 A= with E, or V -- A-type (2-12)
with R -- RX-type (1-12)

is the address of one (if E or R is coded) or two (if V is coded) consecutive fullwords on a fullword boundary. The first word contains the address of the allocated main storage area. If V is coded, the second word contains the length of the main storage area to be released. If R is coded, any of the registers 1 through 12 can be designated, in which case the address of the main storage area to be released, <u>not</u> the address of the fullword, must previously have been loaded into the register. The specification of register 1 saves two bytes in the macro expansion.

FREEMAIN

FREEMAIN -- Release Allocated Main Storage (MVT)

43.1 The FREEMAIN macro instruction releases one or more areas of main storage, or an entire main storage subpool, previously assigned to the active task as a result of a GETMAIN macro instruction. The active task is abnormally terminated if the specified main storage area does not start on a doubleword boundary or if the specified area or subpool is not currently allocated to the active task.

43.2 The control program does not use the main storage area at the address in register 13 as a save area when processing release requests if R is coded.

43.3 The standard form of the FREEMAIN macro instruction is written as shown in the format description below. The operand combinations in the shaded area of the format description below must not be used in an operating system with MFT; the job step would be abnormally terminated.

```
[symbol]  FREEMAIN  { E,LV=number,A=address[,SP=number]      }
                    { L,LA=address,A=address[,SP=number]     }
                    { R,SP=number                            }
                    { R,SP=(0)                               }
                    { R,LV=(0),A=address                     }
                    { R,LV=(0),A=(1)                         }
                    { R,LV=number,A=address[,SP=number]      }
                    { R,LV=number,A=(1)[,SP=number]          }
                    { V,A=address[,SP=number]                }

Note:  only those operand combinations indicated above are valid.
```

43.4 E

(element) written as shown; specifies release of a single area of main storage allocated from the subpool indicated by the SP operand. The length of the main storage area is indicated by the LV operand; the address of the main storage area is provided at the address indicated in the A operand.

43.5 L

(list) written as shown; specifies release of one or more areas of main storage from the subpool indicated by the SP operand. The length of each main storage area is indicated by the values in a list beginning at the address specified in the LA operand. The address of each of the main storage areas must be provided in a corresponding list whose address is specified in the A operand. All main storage areas must start on a doubleword boundary.

43.6 R

(register) written as shown; specifies release of one area of main storage from the subpool indicated by the SP operand, or specifies release of the entire supbool indicated by the SP operand. If the release is not for the entire subpool, the address of the main storage area is indicated by the A operand. The length of the area is indicated by the LV operand. The main storage area must start on a doubleword boundary.

43.7 V

(variable) written as shown; specifies release of one area of main storage from the subpool indicated by the SP operand. The address and length of the main storage area are provided at the address specified in the A operand.

43.8 LV= Sym, Dec Dig, (2-12)

is the length, in bytes, of the main storage area being released. The value should be a multiple of eight; if it is not, the control program uses the next higher multiple of eight. 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 operand is invalid).

43.9 A= with E, L, or V -- A-type, (2-12)
with R -- RX-type, (1-12)

is the main storage address of one or more consecutive fullwords, starting on a fullword boundary. If the words are within an area to be released, they must be completely within the area, and must not begin in the first two words of the first area. If E or R is designated, one word, which contains the address of the main storage area to be released, is required. If V is coded, two words are required; the first word contains the address of the main storage area to be released, and the second word contains the length of the area. If L is coded, one word is required for each main storage area to be released; each word contains the address of one main storage area. If R is coded, any of the registers 1 through 12 can be designated, in which case the address of the main storage area, not the address of the fullword, must have previously been loaded into the register. The specification of register 1 saves two bytes in the macro expansion.

43.10 LA= A-type, (2-12)

is the main storage address of one or more consecutive fullwords starting on a fullword boundary. One word is required for each main storage area to be released; the high-order bit in the last word must be set to one 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 operand. If the words are within an area to be released, they must be completely within the area, and must not begin in the first two words of the first area. The words must not overlap the main storage area specified in the A operand.

43.11 SP= Sym, Dec Dig, (0,2-12)

if the SP operand is optional (shown within brackets), it specifies the subpool number of the main storage area to be released. The subpool number can be between 0 and 127. If the SP operand is optional and is omitted, subpool 0 is assumed. If the SP operand must be coded, it specifies the number of the subpool to be released, and the valid range is 1 through 127. Subpool 0 cannot be released. SP=(0) can be designated, in which case the subpool number must be previously loaded into the high-order byte of register 0; the three low-order bytes must be set to zero.

FREEMAIN - L Form

FREEMAIN -- List Form

44.1 The list form of the FREEMAIN macro instruction is used to construct a control program parameter list. The list and execute forms of the FREEMAIN macro instruction cannot be used with the register (R) type of the macro instruction.

44.2 The description of the standard form of the FREEMAIN macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the list form only. The operands in the shaded area of this format description are used only with MVT; they are ignored if coded with MFT. The L type must not be designated with MFT. The rest of the operands in the nonshaded area can be coded with any configuration of the operating system.

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

[symbol]	FREEMAIN	[E][,LV=number][,A=address][,SP=number] [L][,LA=address][,A=address][,SP=number] [V][,A=address][,SP=number] ,MF=L
Note: only those operand combinations indicated above are valid.		

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

number
: is any absolute expression valid in the assembler language.

MF=L
: indicates the list form of the FREEMAIN macro instruction.

FREEMAIN -- Execute Form

45.1 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 macro instruction. The list and execute forms of the FREEMAIN macro instruction cannot be used with the register (R) type of the macro instruction.

45.2 The description of the standard form of the FREEMAIN macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the execute form only. The operands in the shaded area of this format description are used only with MVT; they are ignored if coded with MFT. The L type must not be designated with MFT. The rest of the operands in the nonshaded area can be coded with any configuration of the operating system.

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

[symbol]	FREEMAIN	[E][,LV=number][,A=address][,SP=number] [L][,LA=address][,A=address][,SP=number] [V][,A=address][,SP=number] ,MF=(E,{control program list address} {(1)})
Note: only those operand combinations indicated above are valid.		

address
: is any address that is valid in an RX-type instruction, or one of general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

number
: is any absolute expression that is valid in the assembler language, or one of general registers 2 through 12, previously loaded with the indicated value. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

MF=(E,{control program list address}
{(1)})
: indicates the execute form of the macro instruction using a remote control program parameter list. The address of the control program parameter list can be coded as described under "address," or can be loaded into register 1, in which case MF=(E,(1)) should be coded.

GETMAIN

GETMAIN -- Allocate Main Storage (MFT)

46.1 The GETMAIN macro instruction is used to allocate an area of main storage for use by the job step task. The main storage is allocated, starting with a doubleword boundary, from the main storage area assigned to the job step. The area is not cleared to zero when allocated. The length of the area requested must not exceed the length available to the job step. The main storage area is released when the job step task terminates or through the use of the FREEMAIN macro instruction.

46.2 The control program does not use the main storage area at the address in register 13 as a save area when processing a request, if R is coded.

46.3 The standard form of the GETMAIN macro instruction is written as shown in the format description below. The operands in the shaded area of the format description are used only in an operating system with MVT. If they are coded in an operating system with MFT and are coded correctly, they are ignored; if they are not coded correctly, the job step is abnormally terminated. The operands in the nonshaded area can be coded with any configuration of the operating system.

```
[symbol] GETMAIN  EC,LV=number,A=address[,SP=number][,HIARCHY=number]
                  EU,LV=number,A=address[,SP=number][,HIARCHY=number]
                  R,LV=number [,SP=number] [,HIARCHY=number]
                  R,LV=(0) [,HIARCHY=number]
                  VC,LA=address,A=address[,SP=number][,HIARCHY=number]
                  VU,LA=address,A=address[,SP=number][,HIARCHY=number]

Note:  only those operand combinations indicated above are valid
```

46.4 E

(element) written as shown; specifies a request for a single area of main storage, with a length indicated by the LV operand. The address of the allocated main storage area is returned at the address indicated in the A operand.

46.5 R

(register) written as shown; specifies a request for a single area of main storage, with a length indicated by the LV operand. The address of the allocated main storage area is returned in register 1. If R is designated, the requests are unconditional; a request for more main storage than is available results in abnormal termination of the job step. If abnormal termination is already in progress, a return code of 4 is placed in register 15.

46.6 V

(variable) written as shown; specifies a request for a single area of main storage, with a length to be between the two values located at the address specified in the LA operand. The address and actual length of the allocated main storage area are returned by the control program at the address indicated in the A operand.

46.7 C

(conditional) written as shown; specifies that the request is conditional and that the job step is not to be abnormally terminated if more main storage area is requested than is available. If the request is satisfied, register 15 contains return code of zero; if not satisfied, the return code is four.

46.8 U

(unconditional) written as shown; specifies that the request is unconditional. An unconditional request for more main storage than is available will result in abnormal termination of the job step. However, if abnormal termination is already in progress, a return code of 4 will be placed in register 15.

46.9 LV= Sym, Dec Dig, (2-12)

is the length, in bytes, of the requested main storage area. The value should be a multiple of eight; if it is not, the control program uses the next higher multiple of eight. If R is coded, LV=(0) may be designated; the low-order three bytes of register 0 must contain the length. The contents of the high-order byte are ignored. (With MVT, the high-order byte contains the subpool number.)

46.10 LA= A-type, (2-12)

is the main storage address of two consecutive fullwords, starting on a fullword boundary. The first word contains the minimum length acceptable; the second word contains the maximum length. The lengths should be multiples of eight; if they are not, the control program uses the next higher multiple of eight.

46.11 A= A-type, (2-12)

is the address of one (if E is coded) or two (if V is coded) consecutive fullwords on a fullword boundary to contain the results of the request. The control program places the address of the allocated main storage area in the first fullword, and, if V is coded, places the length of the main storage area in the second fullword.

46.12 HIARCHY= Dec Dig

specifies the number of the hierarchy from which storage is to be allocated. The number must be 0 to obtain processor storage or 1 to obtain IBM 2361 Core Storage. If the HIARCHY parameter is omitted, HIARCHY=0 is assumed. The main storage will be allocated from the requester's partition segment within the specified hierarchy. However, if a partition is defined entirely within one hierarchy and the request specifies the other hierarchy, the request is defaulted to the hierarchy in which the partition is located. The HIARCHY operand is ignored if main storage hierarchy support is not included in the system.

46.13 After execution of conditional requests, the return code in the low-order byte of register 15 is as follows:

Hex Code	Meaning
0	The main storage requested was allocated.
4	No main storage was allocated.

Note: A request for zero bytes or an unconditional request for more main storage than is available results in abnormal termination of the job step.

GETMAIN -- Allocate Main Storage (MVT)

47.1 The GETMAIN macro instruction requests the control program to allocate one or more areas of main storage to the active task. The main storage areas are allocated from the specified subpool in the main storage area assigned to the associated job step. The main storage areas each begin on a doubleword boundary and are not cleared to zero when allocated. The total of the lengths specified must not exceed the length available to the job step. The main storage areas are released when the task assigned ownership terminates, or through the use of the FREEMAIN macro instruction.

47.2 The control program does not use the main storage area of the address in register 13 as a save area when processing release requests, if R is coded.

47.3 The standard form of the GETMAIN macro instruction is written as shown in the format description below. The operand combinations in the shaded area of the format description below must not be used in an operating system with MFT; the job steps would be abnormally terminated.

[symbol]	GETMAIN	EC,LV=number,A=address [,SP=number][,HIARCHY=number]
		EU,LV=number,A=address [,SP=number][,HIARCHY=number]
		LC,LA=address,A=address[,SP=number][,HIARCHY=number]
		LU,LA=address,A=address[,SP=number][,HIARCHY=number]
		R,LV=number,SP=number [,HIARCHY=number]
		R,LV=(0)[,HIARCHY=number]
		VC,LA=address,A=address[,SP=number][,HIARCHY=number]
		VU,LA=address,A=address[,SP=number][,HIARCHY=number]
Note: only those operand combinations indicated above are valid		

47.4 E

(element) written as shown; specifies a request for a single area of main storage from the subpool indicated by the SP number, having a length indicated by the LV operand. The address of the allocated main storage area is returned at the address indicated in the A operand.

47.5 L

(list) written as shown; specifies a request for one or more areas of main storage from the subpool indicated by the SP number. The length of each main storage area is indicated by the values in a list beginning at the address specified in the LA operand. The address of each of the main storage areas is returned in a list beginning at the address specified in the A operand. No main storage is allocated unless all of the requests in the list can be satisfied.

47.6 R

(register) written as shown; specifies a request for a single area of main storage to be allocated from the indicated subpool, and to have a length indicated by the LV operand. The address of the allocated main storage area is returned in register 1. If R is designated, the requests are unconditional; a request for more main storage than is available results in abnormal termination of the task.

47.7 V

(variable) written as shown; specifies a request for a single area of main storage to be allocated from the subpool indicated by the SP number, and to have a length to be between two values at the address specified in the LA operand. The address and actual length of the allocated main storage area are returned by the control program at the address indicated in the A operand.

47.8 C

(conditional) written as shown; specifies that the request is conditional and that the task is not to be abnormally terminated if more main storage is requested than is available. If the request is satisfied, register 15 contains a return code of zero; if not satisfied, the return code is four.

47.9 U

(unconditional) written as shown; specifies that the request is unconditional. An unconditional request for more main storage than is available will result in abnormal termination of the requesting task.

47.10 LV= Sym, Dec Dig, (2-12)

is the length, in bytes, of the requested main storage area. The number should be a multiple of eight; if it is not, the control program uses the next higher multiple of eight. 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.

47.11 LA= A-type, (2-12)

is the main 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 zero. The lenghts should be multiples of eight; if they are not, the control program uses the next higher multiple of eight. If V was coded, two words are required. The first word contains the minimum length required, the second word contains the maximum length. If L was coded, one word is required for each main storage area requested; the high-order bit in the last word must be set to one to indicate the end of the list. The list must not overlap the main storage area specified in the A operand.

47.12 A= A-type, (2-12)

is the main storage address of consecutive fullwords, starting on a fullword boundary. The control program places the address of the main storage area allocated in the low-order three bytes. If E was coded, one word is required. If L was coded, one word is required for each entry in the LA list. If V was coded, two words are required. The first word contains the address of the main storage area, and the second word contains the length actually allocated. The list must not overlap the main storage area specified in the LA operand.

47.13 SP= Sym, Dec Dig, (2-12)

is the number of the subpool from which the main storage area is to be allocated. The number must be between 0 and 127. If the operand is omitted, subpool zero is specified.

47.14 HIARCHY= Dec Dig

specifies the number of the hierarchy from which storage is to be allocated. The number must be 0 to obtain processor storage or 1 to obtain IBM 2361 Core Storage. If the HIARCHY parameter is omitted, HIARCHY=0 will be assumed. The request will be filled

from the requester's region part within the specified hierarchy. If the request is unconditional and is for more main storage than is available, the task is abnormally terminated. When the unconditional request cannot be satisfied from the requester's region part and rollout/rollin is present, an attempt may be made to obtain storage outside the region part, but within the specified hierarchy.

Note: If only a single hierarchy region exists, all GETMAIN requests will be directed to that hierarchy, regardless of any hierarchy designation in the request. The HIARCHY operand is ignored in an operating system that does not have main storage hierarchy support.

47.15 After execution of conditional requests, the return code in the low-order byte of register 15 is as follows:

Hex Code	Meaning
0	The main storage requested was allocated.
4	No main storage was allocated.

GETMAIN -- List Form

48.1 The list form of the GETMAIN macro instruction is used to construct a control program parameter list. The list and execute forms of the GET-MAIN macro instructions cannot be used with the register (R) type of the macro instruction.

48.2 The description of the standard form of the GETMAIN macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the list form only. The operands in the shaded area of this format description are used only with MVT; they are ignored if coded with MFT. The LC and LU types must not be designated with MFT. The rest of the operands in the nonshaded area can be coded with any control program.

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

[sybmol]	GETMAIN	[EC] [,LV=number] [,A=address] [,SP=number] [EU] [,LV=number] [,A=address] [,SP=number] [LC] [,LA=address] [,A=address] [,SP=number] [LU] [,LA=address] [,A=address] [,SP=number] [VC] [,LA=address] [,A=address] [,SP=number] [VU] [,LA=address] [,A=address] [,SP=number] [,HIARCHY=number] ,MF=L
Note: only those operand combinations indicated above are valid.		

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

number
    is any absolute expression valid in the assembler language.

MF=L
    indicates the list form of the GETMAIN macro instruction.

GETMAIN - E Form

GETMAIN -- Execute Form

49.1 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 macro instruction. The list and execute forms of the GETMAIN macro instruction cannot be used with the register (R) type of the macro instruction. If the GETMAIN macro instruction specifies a remote control program parameter list created by the list form of a FREEMAIN macro instruction, the request will be unconditional unless that specification is replaced by the appropriate conditional operand in the execute form of the GETMAIN macro instruction.

49.2 The description of the standard form of the GETMAIN macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the execute form only. The operands in the shaded area of this format description are used only with MVT; they are ignored if coded with MFT. The LC and LU types must not be designated with MFT. The rest of the operands in the nonshaded area can be coded with any control program.

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

```
[symbol] GETMAIN [EC] [,LV=number] [,A=address] [,SP=number]
                 [EU] [,LV=number] [,A=address] [,SP=number]
                 [LC] [,LA=address] [,A=address] [,SP=number]
                 [LU] [,LA=address] [,A=address] [,SP=number]
                 [VC] [,LA=address] [,A=address] [,SP=number]
                 [VU] [,LA=address] [,A=address] [,SP=number]

                 [,HIARCHY=number]

                 ,MF=(E,{control program list address})
                        {(1)                         }

Note: only those operand combinations indicated above are valid.
```

address
: is any address that is valid in an RX-type instruction, or one of general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

number
: is any absolute expression that is valid in the assembler language, or one of general registers 2 through 12, previously loaded with the indicated value. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

MF=(E,{control program list address} {(1)})
: indicates the execute form of the macro instruction using a remote control program parameter list. The address of the control program parameter list can be coded as described under address, or can be loaded into register 1, in which case MF=(E,(1)) should be coded.

GTRACE -- Record Trace Data

50.1 The GTRACE macro instruction enables a program to have data originating with the program recorded by the Generalized Trace Facility (GTF) in the trace data set identified in the GTF job control statements. The data set must be in the user's partition or region and may later become input to an editing function provided by the IMDPRDMP service aid. GTF must be active and conditioned to accept data originating in a program issuing GTRACE.

50.2 An optional parameter allows specification of a format routine that is to process the record when the trace output is edited by IMDPRDMP. (See the Service Aids publication for GTF and IMDPRDMP details). The GTRACE macro instruction is coded as follows:

[symbol]	GTRACE	DATA=address,LNG=number,ID=value[,FID=number]

DATA= RX-type (2-12)
is the main storage address of the data to be recorded.

LNG= Sym, Dec Dig, (2-12)
is the number of bytes of data to be recorded. Any number from 1 to 256 may be specified.

ID= Sym, Dec Dig
is the identifier to be associated with the record. ID values are assigned as follows:

0-1023	user events.
1024-4095	reserved.

FID= Sym, Dec Dig, (2-12)
indicates the format routine that is to process the record when the trace output is edited using IMDPRDMP. FID values are assigned as follows:

0	entry is to be automatically dumped in hexadecimal.
1-80	user format identifiers.
81-255	reserved.

If the FID parameter is omitted, 0 is assumed.

The format identifier is converted to hexadecimal and, if non-zero, is appended to the name IMDUSR to form the name of the format routine which will be used by IMDPRDMP to process the record.

Formatting routines must be available in SYS1.LINKLIB or in a private library defined in a JOBLIB or STEPLIB DD statement in the JCL for IMDPRDMP.

50.3 GTRACE Macro Instruction Return Codes: Return codes are placed in register 15 when control is returned to the program issuing GTRACE.

Hex Code	Meaning
00	Successful completion.
04	GTF not active or not accepting USR (application program) entries.
08	The length specified in the LNG parameter is greater than 256.
0C	Invalid data address.

GTRACE

14	The value specified in the ID parameter is greater than 1023.
18	Buffers are full, record was not placed in the buffer.
1C	Address of the parameter list is invalid.
10	FID value greater than 255.

GTRACE -- List Form

51.1 The list form of the GTRACE macro instruction constructs a control program parameter list. The description of the standard form provides the explanation of the function of each operand. The format description below indicates the optional and required operands in the list form of the GTRACE macro instruction.

[symbol]	GTRACE	[,DATA=address][,LNG=number][,FID=number],MF=L

address
is any address that is valid in an RX-type instruction.

number
is any absolute expression valid in the assembler language.

MF=L
indicates the list form of the macro instruction

The ID parameter is not valid in the list form of the macro instruction.

GTRACE

GTRACE -- Execute Form

52.1 The execute form of the GTRACE macro instruction uses the remote control program parameter list created by the list form of the macro instruction. The description of the standard form of the macro instruction provides the explanation of the function of each operand. The format description following indicates the optional and required operands for the execute form of the GTRACE macro instruction.

```
[symbol] | GTRACE | ID=value[,DATA=address][,LNG=number][,FID=number],
         |        | MF=(E,{parameter list address})
         |        |       {      (1-12)          }
```

address
: is any address that is valid in an RX-type instruction, or one of the general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

number
: is any absolute expression that is valid in the assembler language, or one of the general registers 2 through 12, previously loaded with the indicated value. The register may be designated symbolically or within an absolute expression, and is always coded within parentheses.

value
: is any absolute expression valid in the assembler language.

MF=(E,{parameter list address}) / {(1-12)}
: indicates the execute form of the macro instruction using a remote parameter list. The address of the parameter list can be loaded into register 1, in which case MF=(E,(1)) should be coded. If the address is not loaded into register 1, it can be coded as any address that is valid in an RX-type instruction, or one of the general registers 2-12, previously loaded with the address. A register can be designated symbolically or with an absolute expression, and is always coded within parentheses.

IDENTIFY -- Add an Entry Point (MFT Without Identify Option)

53.1 The identify function is an optional feature of an operating system with MFT. If the identify function was not selected when the operating system was generated, the use of an IDENTIFY macro instruction results in an effective NOP instruction to provide compatibility with an operating system that does include the function.

53.2 The IDENTIFY macro instruction is written as follows:

```
[symbol] | IDENTIFY | {EP=symbol                }
         |          | {EPLOC=address of name    },ENTRY=entry point address
```

The contents of register 15 are set to zero.

IDENTIFY -- Add an Entry Point (MFT With Identify Option, MVT)

54.1 The IDENTIFY macro instruction is used to add an entry point to a copy of a load module currently in main storage. The load module copy must be one of the following:

- A copy that satisfied the requirements of a LOAD macro instruction issued during the performance of the same task.
- In an MFT system with subtasking, a copy that satisfied the requirements of a LOAD macro instruction issued during the performance of any task within the partition.
- The last load module given control, if control was passed to the load module using a LINK, ATTACH, or XCTL macro instruction.
- The first load module of the job step, if it is still in control.
- In an MFT system with subtasking, the first load module of any task, if it is still in control.

54.2 The IDENTIFY macro instruction may not be issued by an asynchronous exit routine or a synchronous exit routine entered via SYNCH processing; the routine associated with the entry point is assumed to be reenterable.

- MFT: The IDENTIFY macro instruction may not be issued by a routine entered at an added entry point. The added entry point can be used only in an ATTACH macro instruction.
- MVT: The added entry point can be used in an ATTACH, LINK, LOAD, DELETE, or XCTL macro instruction.

54.3 The IDENTIFY macro instruction is written as follows:

```
┌─────────┬─────────┬──────────────────────────────────────────────────────┐
│[symbol] │IDENTIFY │⎧EP=symbol            ⎫,ENTRY=entry point address    │
│         │         │⎩EPLOC=address of name⎭                              │
└─────────┴─────────┴──────────────────────────────────────────────────────┘
```

EP= Symbol

is the name of the entry point. The name does not have to correspond to any name or symbol in the load module, and must not correspond to any name, alias, or added entry point for a load module in the link pack area or the job pack area of the jobstep.

EPLOC= RX-type, (0,2-12)

is the address of the entry point name described under EP. The name must be padded to the right with blanks to eight bytes, if necessary.

ENTRY= RX-type, (1-12)

is the main storage address of the entry point being added.

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

Hex Code	Meaning
00	Successful completion.
04	Entry point name and address already exist.
08	Entry point name duplicates the name of a load module currently in main storage; entry point was not added.
0C	Entry point address is not within an eligible load module; entry point was not added.
10	Issued by an asynchronous exit routine or by a synchronous exit routine entered via SYNCH; the entry point was not added.
14	An IDENTIFY macro instruction was previously issued using the same entry point name but a different address; this request was ignored.

LINK -- Pass Control to a Program in Another Load Module

55.1 The LINK macro instruction causes control to be passed to a specified entry point; the entry point name must be a member name or an alias in a directory of a partitioned data set. (With MVT, the entry point can be an added entry point specified in an IDENTIFY macro instruction.) The load module containing the program is brought into main storage if a useable copy is not available. (See Section I for a discussion of the use of an existing copy of the load module.)

55.2 The linkage relationship established is the same as that created by a BAL instruction; control is returned to the instruction following the LINK macro instruction after execution of the called program. The problem program optionally can provide a parameter list to be passed to the called program. If the called program terminates abnormally, or if the specified entry point cannot be located, the task is abnormally terminated.

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

```
┌─────────┬─────────┬──────────────────────────────────────────────────────┐
│[symbol] │  LINK   │ ⎧EP=symbol                  ⎫[,DCB=dcb address]       │
│         │         │ ⎨EPLOC=address of name      ⎬                         │
│         │         │ ⎩DE=address of list entry   ⎭                         │
│         │         │                                                      │
│         │         │ [,PARAM=(addresses)[,VL=1]]                          │
│         │         │                                                      │
│         │         │ [,ID=number][,HIARCHY=number]                        │
└─────────┴─────────┴──────────────────────────────────────────────────────┘
```

55.4 EP= Sym

is the entry point name in the program to be given control.

55.5 EPLOC= A-type, (2-12)

is the address of the entry point name described above. The name must be padded with blanks to eight bytes, if necessary.

55.6 DE= A-type, (2-12)

is the address of the name field of a list entry for the entry point name. The list entry is constructed using the BLDL macro instruction. The DCB operand must indicate the same data control block used in the BLDL macro instruction. If the module is indicated as being in the job, step, or task library by the Z byte of the BLDL list entry, the LINK macro instruction must be either in the same task as the BLDL or in a task with the same chain of task libraries.

55.7 DCB= A-type, (2-12)

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

If the DCB= operand is omitted or if DCB=0 is specified when the LINK macro instruction is issued by the job step task, the data sets referenced by either the STEPLIB or JOBLIB DD statement are first searched for the entry point name. If the entry point name is not found, the link library is searched.

If the DCB= operand is omitted or if DCB=0 is specified when the LINK macro instruction is issued by a subtask, the data set(s) associated with one or more data control blocks referenced by previous ATTACH macro instructions in the subtasking chain are first searched for the entry point name. If the entry point name is not found, the search is continued as if the LINK macro instruction had been issued by the job step task.

55.8 PARAM= A-type, (2-12)

is one or more address parameters, separated by commas, to be passed to the called program. Each address is expanded in line to a fullword on a fullword boundary, in the order designated. Register 1 contains the address of the first parameter when the program is given control. (If this operand is omitted, register 1 is not altered.)

55.9 VL=1

is written as shown. It can be designated only if PARAM is designated, and should be used only if the called program can be passed a veriable number of parameters. VL=1 causes the high-order bit of the last address parameter to be set to 1; the bit can be checked to find the end of the list.

55.10 ID= Sym, Dec Dig

maximum value is 2^{16}-1. The last fullword of the macro expansion is a NOP instruction containing the ID value in the low-order two bytes.

55.11 HIARCHY= Dec Dig

specifies the storage hierarchy (0 or 1) in which the load module is to be loaded when a usable copy is not already available in main storage. If the HIARCHY parameter is missing, loading will take place according to the hierarchy specified at Link Edit time. If HIARCHY is specified, it will override any hierarchy assignments made during linkage editing. The HIARCHY operand is ignored in an operating system that does not have main storage hierarchy support.

LINK - L Form

LINK -- List Form

56.1 Two parameter lists are used in a LINK macro instruction: a control program parameter list and an optional problem program parameter list. Only the control program parameter list can be constructed in the list form of the LINK macro instruction. 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 the LINK macro instruction.

56.2 The description of the standard form of the LINK macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the list form only.

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

```
[symbol]   LINK    ⎡EP=symbol                 ⎤[,DCB=dcb address]
                   ⎢EPLOC=address of name     ⎥
                   ⎣DE=address of list entry  ⎦
                   [,HIARCHY=number],SF=L
```

symbol
    is any symbol valid in the assembler language.

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

SF=L
    indicates the list form of the LINK macro instruction.

LINK -- Execute Form

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

57.2 The description of the standard form of the LINK macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are completely optional and which are required in at least one of a pair of list and execute forms. The format description below indicates the optional and required operands in the execute form only.

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

```
[symbol] | LINK | ⎡EP=symbol              ⎤[,DCB=dcb address]
         |      | ⎢EPLOC=address of name  ⎥
         |      | ⎣DE=address of list entry⎦
         |      | [,HIARCHY=number][,PARAM=(addresses)[,VL=1]]
         |      | [,ID=number]
         |      |
         |      | ⎧,MF=(E,{problem program list address})       ⎫
         |      | ⎪       {   (1)                      }        ⎪
         |      | ⎨,SF=(E,{control program list address})       ⎬
         |      | ⎪       {   (15)                     }        ⎪
         |      | ⎪,MF=(E,{address}),SF=(E,{address})           ⎪
         |      | ⎩       {  (1)  }        {  (15) }            ⎭
```

57.4 symbol
is any symbol valid in the assembler language.

57.5 address
is any address that is valid in an RX-type instruction, or one of general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

57.6 number
is any absolute expression that is valid in the assembler language.

57.7 MF=(E,{problem program list address | (1)})
indicates the execute form of the macro instruction using a remote problem program parameter list. Any control program parameters specified are provided in a control program parameter list expanded in line. The address of the problem program parameter list can be coded as described under address, or can be loaded into register 1, in which case MF=(E,(1)) should be coded.

57.8 SF=(E,{control program list address | (15)})
indicates the execute form of the macro instruction using a remote control program parameter list. Any problem program parameters specified are provided in a problem program parameter list expanded

in line. The address of the control program parameter list can be coded as described under "address," or can be loaded into register 15, in which case SF=(E,(15)) should be coded.

57.9 MF=(E,{address}),SF=(E,{address})
{ (1) } { (15) }

indicates the execute form of the macro instruction using both a remote problem program parameter list and a remote control program parameter list. The addresses of the parameter lists are coded or loaded into registers 1 and 15, as explained above.

LOAD -- Bring a Load Module Into Main Storage

58.1 The LOAD macro instruction causes the control program to bring the load module containing the specified entry point into main storage, if a usable copy is not available in main storage. (See Section I for a discussion of the use of an existing copy of the load module.) The responsibility count for the load module is increased by one. Control is not passed to the load module; instead, the main storage address of the designated entry point is returned in register 0. The load module remains in main storage until the responsibility count is reduced to zero through task terminations or through the use of the DELETE macro instruction.

58.2 The entry point name in the load module must be a member name or an alias in a directory of a partitioned data set. In an operating system with MVT, the name can also be an added entry point specified in an IDENTIFY macro instruction. If the specified entry point cannot be located, the task is abnormally terminated.

58.3 The LOAD macro instruction is written as follows:

```
[symbol] | LOAD | {EP=symbol                  } [,DCB=dcb address]
         |      | {EPLOC=address of name      }
         |      | {DE=address of list entry   }
         |      |
         |      |  [,HIARCHY=number]
```

58.4 EP= Symbol
is the entry point name in the load module to be brought into main storage.

58.5 EPLOC= RX-type, (0,2-12)
is the main storage address of the entry point name described above. The name must be padded with blanks to eight bytes, if necessary.

58.6 DE= RX-type, (0,2-12)
is the address of the name field of a list entry for the entry point name. The list entry is constructed by a BLDL macro instruction. The DCB operand must indicate the same data control block used in the BLDL macro instruction. If the module is indicated as being in the job, step, or task library by the Z byte of the BLDL list entry, the LOAD macro instruction must be either in the same task as the BLDL or in a task with the same chain of task libraries

58.7 DCB= RX-type, (1-12)
is the address of the data control block for the partitioned data set containing the entry point name described above.

If the DCB= operand is omitted or if DCB=0 is specified when the LOAD macro instruction is issued by the job step task, the data sets referenced by either the STEPLIB or JOBLIB DD statement are first searched for the entry point name. If the entry point name is not found, the link library is searched.

If the DCB= operand is omitted or if DCB=0 is specified when the LOAD macro instruction is issued by a subtask, the data set(s) associated with one or more data control blocks referenced by previous ATTACH macro instructions in the subtasking chain are first searched for the entry point name. If the entry point name is not found, the search is continued as if the LOAD macro instruction had been issued by the job step task.

58.8 HIARCHY= Dec Dig

specifies the storage hierarchy (0 or 1) in which the load module is to be loaded. If the HIARCHY parameter is missing, loading will take place according to the hierarchy specified at Link Edit time. If HIARCHY is specified, it will override any hierarchy assignments made during linkage editing. The HIARCHY operand is ignored in an operating system that does not have main storage hierarchy support.

POST -- Signal Event Completion

59.1 The POST macro instruction causes the specified event control block (ECB) to be set to indicate the occurrence of an event. This event satisfies the requirements of a WAIT macro instruction if this is the last or only event specified by the WAIT. If more events are awaited, the WAIT remains unsatisfied. The bits in the ECB are set as follows:

Bit 0 of the specified ECB is set to 0.

Bit 1 of the specified ECB is set to 1.

Bits 8 through 31 are set to the specified completion code value.

59.2 The POST macro instruction is written as follows:

[symbol]	POST	ecb address[,completion code]

ecb address (1-12)
: is the address of an event control block representing the event.

completion code (0,2-12)
: value between 0 and $2^{24}-1$. If the completion code is not designated, 0 is assumed.

RETURN

RETURN -- Return Control

60.1 The RETURN macro instruction is used to return control to the calling program and to signal normal termination of the returning program. The return of control is always made by executing a branch instruction using the address in register 14. The RETURN macro instruction can be written to restore a designated range of registers, provide the proper return code in register 15, and flag the save area used by the returning program.

60.2 The RETURN macro instruction is written as follows:

```
┌─────────┬─────────┬────────────────────────────────────────────────────────────┐
│[symbol] │ RETURN  │   [(reg1[,reg2])][,T]┌,RC=number┐                          │
│         │         │                      └,RC=(15)  ┘                          │
└─────────┴─────────┴────────────────────────────────────────────────────────────┘
```

reg1,reg2 — Dec Dig

is the range of registers to be restored from the save area pointed to by the address in register 13. The registers should be designated to cause the loading of registers 14, 15, 0 through 12 (in that order) when used in a LM instruction. If reg2 is not designated, only the register designated by the reg1 operand is loaded. If the operand is omitted, the contents of the registers are not altered. Do not code reg1 or reg2 when returning control from a program interruption exit routine.

T

causes the control program to flag the save area used by the returning program. A byte containing all ones is placed in the high-order byte of word four of the save area after the registers have been loaded. This operand cannot be designated when returning control from an exit routine.

RC= — Sym, Dec Dig (15)

is the return code to be passed to the calling program. The return code should have a maximum value of 4095; it will be placed right-adjusted in register 15 before return is made. If RC=(15) is coded, it indicates that the return code has been previously loaded into register 15; in this case the contents of register 15 are not altered or restored from the save area. (If this operand is omitted, the contents of register 15 are determined by the reg1, reg2 operands.)

SAVE -- Save Register Contents

61.1 The SAVE macro instruction causes the contents of the specified registers to be stored in the save area at the address contained in register 13. An entry point identifier can optionally be specified. The SAVE macro instruction should be written only at the entry point of a program because the code resulting from the macro expansion requires that register 15 contain the address of the SAVE macro instruction. Do not use the SAVE macro instruction in a program interruption exit routine.

61.2 The SAVE macro instruction is written as follows:

[symbol]	SAVE	(reg1[,reg2]),[T][,identifier name]

61.3 reg1,reg2 — Dec Dig

is the range of registers to be stored in the save area at the address contained in register 13. The registers should be designated so they are stored in the order 14, 15, 0 through 12 when used directly in an STM instruction. The registers are stored in words 4 through 18 of the save area. If only one register is designated, only that register is saved.

61.4 T

specifies that registers 14 and 15 are to be stored in words 4 and 5, respectively, of the save area. If both T and reg2 are designated and reg1 is any of registers 14, 15, 0, 1, or 2, all of registers 14 through the reg2 value are saved.

61.5 identifier name

is an identifier to be associated with the SAVE macro instruction. The name may be up to 70 characters and may be a complex name. If an asterisk is coded, the identifier is the symbol associated with the SAVE macro instruction, or, if the name field is blank, the control section name. If the CSECT instruction name field is blank, the operand is ignored. Whenever a symbol or an asterisk is coded, the following macro expansion occurs:

- A count byte, containing the number of characters in the identifier name, is constructed four bytes following the address contained in register 15.
- The character string containing the identifier name is constructed, starting at five bytes following the address contained in register 15.

SEGLD -- Load Overlay Segment and Continue Processing (MFT)

62.1 When used in an operating system with MFT, the SEGLD macro instruction results in an effective NOP instruction to assure compatibility with an operating system with MVT. The SEGLD instruction is written as follows:

[symbol]	SEGLD	external segment name

SEGLD -- Load Overlay Segment and Continue Processing (MVT)

63.1 The SEGLD macro instruction causes the control program to load the specified segment and any segments in its path that are not part of a path already in main storage. Control is not passed to the specified segment, but is returned to the instruction following the SEGLD macro instruction. Processing is overlapped with the loading of the segment. Refer to the Linkage Editor and Loader, for details on overlay operations.

63.2 The SEGLD macro instruction cannot be used in an asynchronous exit routine. The SEGLD macro instruction is written as follows:

[symbol]	SEGLD	external segment name

external segment name — Sym

is the name of a control section or an entry point in the required segment. An exclusive reference is not allowed. The name does not have to be identified by an EXTRN statement.

SEGWT

SEGWT -- Load Overlay Segment and Wait

64.1 The SEGWT macro instruction causes the control program to load the specified segment and any segments in its path that are not part of a path already in main storage. Control is not passed to the specified segment; control is not returned to the segment issuing the SEGWT macro instruction until the requested segment is loaded. Refer to the Linkage Editor and Loader, for details on overlay operations. The SEGWT macro instruction cannot be used in an asynchronous exit routine.

64.2 The SEGWT macro instruction is written as follows:

[symbol]	SEGWT	external segment name

external segment name — Sym

is the name of a control section or entry point in the required segment. An exclusive reference is not allowed. The name does not have to be identified by an EXTRN statement.

SNAP -- Dump Main Storage and Continue (MFT)

65.1 The SNAP macro instruction causes the control program to dump some or all of the main storage areas assigned to the current job step. Some or all of the control program fields can also be dumped. The format of the dump is similar to the abnormal termination dump shown in the Programmer's Guide to Debugging.

65.2 A data set must be supplied to contain the dump. An open data control block must be supplied for the data set, and must contain the following operands:

DSORG=PS,RECFM=VBA,MACRF=(W),BLKSIZE=882,LRECL=125,
and DDNAME=any name but SYSABEND or SYSUDUMP

65.3 The standard form of the SNAP macro instruction is written as shown in the format description below. The operand in the shaded area of the format description is used only in an operating system with MVT; it is ignored if coded in an operating system with MFT. The operands in the nonshaded area can be coded with any configuration of the operating system.

```
[symbol]   SNAP   DCB=dcb address[,TCB=address]
                  [,ID=number] ,SDATA=(code for control
                                     program blocks)
                  [,PDATA=(code for problem program areas)]
                   ,STORAGE=(starting address, ending address,...)
                   ,LIST=address of list
```

65.4 DCB= A-type, (2-12)
the address of the data control block for the data set to contain the dump.

65.5 ID= Sym, Dec Dig (2-12)
a number between 1 and 255. The number is printed in the identification heading associated with the dump.

65.6 SDATA=
one to four of the following sets of characters, written in any order and separated by commas. The characters are used to request the associated control program information. If the SDATA operand is incorrectly coded, the control program assigns the code CB.

Code	Fields Dumped
ALL	All of the following fields.
NUC	All of the control program nucleus except the trace table.
TRT	Trace Table.
CB	Task control block (TCB), active request blocks (RBs), job pack area control queue (JPACQ), and main storage supervisor (MSS) control blocks.
Q	Ignored in an operating system without MVT.

65.7 PDATA=
one to five of the following sets of characters, written in any order and separated by commas. The characters are used to request the associated problem program information. If the PDATA operand is incorrectly coded, the control program assigns the code ALL.

Code	Fields Dumped
ALL	All of the following fields.
PSW	Program Status Word when the SNAP macro instruction was issued.
REGS	Contents of the general registers when the SNAP macro instruction was issued.
SA or SAH	SA - provides linkage information and a back trace through save areas. SA is selected if ALL is coded. SAH - only linkage information.
JPA or LPA or ALLPA	JPA - all main storage assigned to the job step. (Differs in MVT.) LPA - contents of the resident reenterable load module area. (Differs in MVT.) ALLPA - contents of both pack areas. ALLPA is selected if ALL is coded.
SPLS	All main storage assigned to job step. (Differs in MVT.)

65.8 STORAGE= A-type, (2-12)

is one or more pairs of starting and ending addresses; the areas defined by the starting and ending addresses (inclusive) are dumped one fullword at a time. If the starting and ending addresses are not fullword multiples, the addresses are rounded down or up, respectively, to a fullword.

The starting and ending addresses must be in the partition within which the user is operating. If the addresses are not within the partition, the storage area requested is not dumped.

65.9 LIST= A-type, (2-12)

the address of a list of starting and ending addresses of areas to be dumped. The addresses in the list are treated in the same manner as the addresses described under "STORAGE=." The list must begin on a fullword boundary; each address in the list occupies one fullword. The high-order byte of each word containing the starting address of an area to be dumped must contain zeros or that pair will be skipped. The high order bit (bit 0) of the fullword containing the last ending address in the list must be set to 1.

95.10 Control is returned to the instruction following the SNAP macro instruction. When control is returned, register 15 contains one of the following return codes:

Hex Code	Meaning
00	Successful completion.
04	Data control block was not open.
08	Task control block address was not valid (used only with MVT).
0C	Data control block type was not correct (DSORG, RECFM, MACRF, BLKSIZE, or LRECL field).

SNAP -- Dump Main Storage and Continue (MVT)

66.1 The SNAP macro instruction causes the control program to dump some or all of the main storage areas assigned to a task in the current job step. Some or all of the control program fields can also be dumped. The format of the dump is similar to the abnormal termination dump shown in the publication *Programmer's Guide to Debugging*.

66.2 A data set must be supplied to contain the dump. An open data control block must be supplied for the data set, and must contain the following operands:

DSORG=PS,RECFM=VBA,MACRF=(W),BLKSIZE=nnn,LRECL=125,
and DDNAME=any name but SYSABEND or SYSUDUMP.

Where the block size nnn is either 882 or 1632 for MVT.

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

```
┌─────────┬─────────┬───────────────────────────────────────────────────┐
│[symbol] │  SNAP   │ DCB=dcb address[,TCB=address]                     │
│         │         │ [,ID=number] ┌,SDATA=(code for control ┐          │
│         │         │              └          program blocks)┘          │
│         │         │ [,PDATA=(code for problem program areas)]         │
│         │         │ ┌,STORAGE=(starting address, ending address,...)┐ │
│         │         │ └,LIST=address of list                          ┘ │
└─────────┴─────────┴───────────────────────────────────────────────────┘
```

66.4 DCB= — A-type, (2-12)

is the address of the data control block for the data set to contain the dump.

66.5 TCB= — A-type, (2-12)

specifies the address of a fullword on a fullword boundary containing the address of the task control block for a task of the current job step other than the active task. If omitted, or if the fullword contains zero, the dump is for the active task. If a register is designated, the register can contain zero to indicate the active task, or can contain the address of a task control block for a task other than the active task.

66.6 ID= — Sym, Dec Dig, (2-12)

a number between 0 and 255. The number is printed in the identification heading associated with the dump. If the number specified is not among the numbers 0 to 255, then it will not be printed properly in the identification heading.

66.7 SDATA

one to four of the following sets of characters written in any order and separated by commas. The characters are used to request the associated control program information. If the SDATA operand is incorrectly coded, the control program assigns the code CB.

Code	Fields Dumped
ALL	All of the following fields.
NUC	All of the control program nucleus except the trace table.
TRT	Trace table.
CB	Task control block (TCB), active request blocks (RBs), contents directory entry (CDE), load list elements (LLEs), extent list (XL), data extent block (DEB), task

input/output table (TIOT), and main storage supervisor (MSS) control blocks.

Q	Queue control blocks and queue elements.

66.8 PDATA=

is one to five of the following sets of characters written in any order and separated by commas. The characters are used to request the associated problem program information. If the PDATA operand is incorrectly coded, the control program assigns the code ALL.

Code	Fields Dumped
ALL	All of the following fields.
PSW	Program Status Word when the SNAP macro instruction was issued.
REGS	Contents of the general registers when the SNAP macro instruction was issued.
SA or SAH	SA - provides linkage information and a back trace through save areas. SA is selected if ALL is coded. SAH - only linkage information.
JPA or LPA or ALLPA	JPA - contents of the job pack area. LPA - contents of the link pack area. ALLPA - contents of both pack areas, ALLPA is selected if ALL is coded.
SPLS	All main storage subpools (0-127).

66.9 STORAGE= A-type, (2-12)

is one or more pairs of starting and ending addresses; the areas defined by the starting and ending addresses (inclusive) are dumped one fullword at a time. If the starting and ending addresses are not fullword multiples, the addresses are rounded down or up, respectively, to a fullword.

The starting and ending addresses must be within the region within which the user is operating. If the addresses are not within the region, the storage area requested is not dumped.

66.10 LIST= A-type, (2-12)

is the address of a list of starting and ending addresses of areas to be dumped. The addresses in the list are treated in the same manner as the addresses described under STORAGE. The list must begin on a fullword boundary; each address in the list occupies one fullword. The high-order byte of the word containing the starting address of the area to be dumped must contain zeros or that pair will be skipped. The high-order bit (bit 0) of the fullword containing the last ending address in the list must be set to 1.

66.11 Control is returned to the instruction following the SNAP macro instruction. When control is returned, register 15 contains one of the following return codes:

Hex Code	Meaning
00	Successful completion.
04	Data control block was not open.
08	Task control block address was not valid.
0C	Data control block type was not correct (DSORG, RECFM, MACRF, BLKSIZE, or LRECL field).

SNAP -- List Form

67.1 The list form of the SNAP macro instruction is used to construct a control program parameter list. Any number of main storage addresses can be specified using the STORAGE operand. Therefore, the number of starting and ending address pairs in the list form of the SNAP macro instruction must be equal to the maximum number of addresses specified in any execute form of the macro instruction, or a DS instruction must immediately follow the list form to allow for the maximum number of addresses.

67.2 The description of the standard form of the SNAP macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the list form only.

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

```
[symbol]  | SNAP | [DCB=address][,ID=number][,SDATA=(code)]
          |      | [,PDATA=(code)]
          |      |
          |      | [,STORAGE=(address,address,...)] ,MF=L
          |      | [,LIST=address                 ]
```

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

code
: is written as indicated in the description of the standard form of the macro instruction.

number
: is any absolute expression valid in the assembler language.

MF=L
: indicates the list form of the SNAP macro instruction.

SNAP - E Form

SNAP -- Execute Form

68.1 A remote control program parameter list is referred to and can be modified by the execute form of the SNAP macro instruction.

68.2 If only the DCB, ID, MF, or TCB operands are coded in the execute form of the macro instruction, the bit settings in the parameter list corresponding to the SDATA, PDATA, LIST, and STORAGE operands are not changed. However, if one or more of the SDATA, PDATA, LIST operands are coded, the bit settings from the previous request are reset, and only the areas requested in the current macro instruction are dumped.

68.3 The description of the standard form of the SNAP macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the execute form only. The operands in the shaded area of this format description are used only with MVT; they are ignored if coded with MFT. The operands in the nonshaded area can be coded with any control program.

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

```
[symbol]   SNAP   [DCB=address][,TCB={ 'S'    }][,ID=number]
                                      {address}
                  [,PDATA=code][,SDATA=code]
                  [,STORAGE=(address,address,...)]
                  [,LIST=address                 ]
                   ,MF=(E,{control program list address})
                          {(1)                          }
```

address
is any address that is valid in an RX-type instruction, or one of general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

'S'
is used to specify the task control block of the active task.

number
is any absolute expression that is valid in the assembler language, or one of general registers 2 through 12, previously loaded with the indicated value. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

code
is written as indicated in the description of the standard format of the macro instruction.

MF=(E,{control program list address})
{(1)}
indicates the execute form of the macro instruction using a remote control program parameter list. The address of the control program parameter list can be coded as described under "address," or can be loaded into register 1, in which case MF=(E,(1)) should be coded.

<u>Note</u>: Any values (codes or addresses specified by the PDATA, LIST, SDATA, or STORAGE operands remain in effect one of these operands is specified in the execute form. In this case, all values specified by those operands in the list form are canceled, and only those values specified in the execute form remain in effect.

SPIE

SPIE -- Specify Program Interruption Exit

69.1 The SPIE macro instruction is used to specify the address of an interruption exit routine and to specify the program interruption types that are to cause the exit routine to be given control. If the program interruption types specified can be masked, the corresponding program mask bit in the Program Status Word is set to 1.

69.2 The effect of each SPIE macro instruction issued in performance of a task supersedes the effect of the previous SPIE macro instruction issued in performance of the same task. The specified exit routine is given control when one of the specified program interruptions occurs in any program of the task.

69.3 A program interruption control area (PICA) is created as part of the expansion of the SPIE macro instruction. The PICA, shown in Figure 64, contains the exit routine address and a code indicating the interruption types specified in the SPIE macro instruction. The previous PICA address is returned in register 1 after execution of the SPIE macro instruction; this address can be used to restore the PICA before returning control. If no SPIE environment exists when the SPIE macro instruction is issued, register 1 contains zero when control is returned.

69.4 The effect of the last SPIE macro instruction issued is canceled by issuing a SPIE macro instruction with no operands. This action <u>does not</u> reestablish the effect of the previous SPIE. To reestablish a previous SPIE, whether or not a "cancel" SPIE has been issued, an execute form of the SPIE macro instruction may be issued specifying the address of the appropriate PICA. Note that issuing a "cancel" SPIE also causes the address of the previous PICA to be returned (see Section I "Program Interruption Control Area" for other programming considerations).

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

[symbol]	SPIE	[interruption exit address,(interruptions)]

69.6 interruption exit address — A-type, (2-12)

is the address of the exit routine to be given control after a program interruption of the type specified in the "interruptions" operand.

69.7 interruptions — Dec Dig

is one or more decimal numbers, separated by commas, indicating the corresponding interruption type shown below. The interruption types can be designated in any order as follows:

- One or more single numbers, each indicating the corresponding program interruption type.
- One or more pairs of decimal numbers, each pair indicating a range of corresponding interruption types. The second number must be higher than the first. The pair of numbers must be separated from each other by commas and enclosed in an additional set of parentheses.

69.8 For example, (4,8) indicates interruption types 4 and 8; ((4,8)) indicates interruption types 4 through 8, inclusively. 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

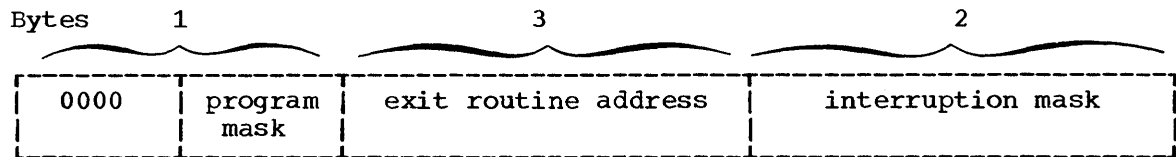


Figure 64. Program Interruption Control Area

SPIE - L Form

SPIE -- List Form

70.1 The list form of the SPIE macro instruction is used to construct a control program parameter list in the form of a program interruption control area.

70.2 The description of the standard form of the SPIE macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the list form only.

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

[symbol]	SPIE	[interruption exit address][,(interruptions)],MF=L

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

interruptions
: are one or more decimal digits separated by commas.

MF=L
: indicates the list form of the SPIE macro instruction.

SPIE -- Execute Form

71.1 A remote control program parameter list (program interruption control area) is used in, and can be modified by, the execute form of the SPIE macro instruction. The program interruption control area can be generated by the list form of the SPIE macro instruction, or the address of the program interruption control area returned in register 1 following a previous SPIE macro instruction can be used. The execute form can be used to reestablish a previously canceled SPIE by specifying that PICA in the MF operand. Both the standard and execute forms cause the address of the previous PICA to be returned in register 1.

71.2 The description of the standard form of the SPIE macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the execute form only. If the address of a previous program interruption control area is used, only the MF operand should be coded.

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

```
┌─────────┬─────────┬──────────────────────────────────────────────────────┐
│[symbol] │  SPIE   │ [interruption exit address][,(interruptions)]        │
│         │         │                                                      │
│         │         │ ,MF=(E,{control program list address})               │
│         │         │        {(1)                          }               │
└─────────┴─────────┴──────────────────────────────────────────────────────┘
```

address
: is any address that is valid in an RX-type instruction, or one of general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

interruptions
: are one or more decimal numbers separated by commas.

MF=(E, control program list address)
       (1)
: indicates the execute form of the macro instruction using a remote control program parameter list (program interruption control area). The address of the control program parameter list can be coded as described under address, or can be loaded into register 1, in which case MF=(E,(1)) should be coded.

72.1

STAE

STAE -- Specify Task Abnormal Exit

72.1 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.

72.2 The STAE macro instruction creates a STAE control block (SCB) which represents a STAE environment that remains in effect during the execution of the program that issued the STAE or until canceled by a subsequent STAE. When a RETURN, XCTL, or SVC 3 is issued, the system automatically cancels the STAE environment for that program, unless XCTL=YES is coded in the STAE macro instruction. If XCTL=YES is coded and an XCTL macro instruction is issued, the STAE environment remains in effect for the program that receives control as a result of the XCTL macro instruction.

72.3 When a STAE environment is canceled, the last STAE environment that was created and not subsequently overlayed or canceled (if any) becomes the current STAE environment.

72.4 Note that issuing a LINK macro instruction does not cancel the STAE environment and that the user is responsible for canceling the STAE environment if his program does not exit via a RETURN, XCTL, or SVC3. The user cannot cancel or overlay a STAE control block not created by his own program.

72.5 Within the STAE exit routine, the user may perform pre-termination functions or diagnose an error. Upon completion of STAE exit routine processing, the user can either allow abnormal termination processing to continue for the task or request that a STAE retry routine be scheduled which would circumvent the scheduled ABEND. For further explanation of the facility for scheduling a STAE retry routine, see the _MFT Guide_ or the _MVT Guide_.

72.6 The STAE exit routine cannot contain a STAE or an ATTACH macro instruction. When a STAE retry routine is not to be scheduled, the STAE exit routine should return with a code of zero in register 15.

72.7 Entry to a STAE retry routine cancels the STAE environment. If a STAE retry routine causes the task to resume execution, the STAE environment should be reestablished from within the retry routine.

72.8 The STAE macro instruction is written as follows:

```
┌─────────┬─────────┬──────────────────────────────────────────────────────┐
│         │         │ {      0       } [,OV]                                │
│[symbol] │  STAE   │ {exit address  } [,CT][,PARAM=list address]           │
│         │         │                                                       │
│         │         │ [       {YES}] [        {QUIESCE}]                    │
│         │         │ [,XCTL={NO }] [,PURGE={HALT    }]                     │
│         │         │                [        {NONE   }]                    │
│         │         │                                                       │
│         │         │ [        {YES}]                                       │
│         │         │ [,ASYNCH={NO }]                                       │
└─────────┴─────────┴──────────────────────────────────────────────────────┘
```

(Defaults underlined on the page: CT, NO, QUIESCE, NO.)

72.9 exit address A-type, (2-12)

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. The address may be loaded into one of the general registers 2 through 12.

72.10 OV

indicates that the parameters passed in this STAE macro instruction are to overlay the data contained in the previous STAE request. In the standard form only of the STAE macro instruction, if any of the parameters XCTL, PURGE, or ASYNCH are not specified, the default value for the omitted parameter is assigned.

72.11 CT

indicates the creation of a new STAE request. If neither OV or CT is specified, CT is assumed.

72.12 PARAM= A-type, (2-12)

specifies the address of a parameter list containing data to be used by the STAE exit routine when it is scheduled for execution. The address may be loaded into one of the general registers 2 through 12.

72.13 XCTL=YES

indicates that the STAE macro instruction will not be canceled if an XCTL macro instruction is issued.

72.14 XCTL=NO

indicates that the STAE macro instruction will be canceled if an XCTL is issued by this program. If neither XCTL=YES or XCTL=NO is coded, XCTL=NO is assumed.

72.15 PURGE=

QUIESCE

indicates that all outstanding requests for input/output (I/O) operations will be saved when the STAE exit is taken. At the end of the STAE exit routine, the user can code a retry routine to handle the outstanding I/O requests. (See the description of the STAE macro instruction in the MFT Guide or the MVT Guide for a description of the STAE retry routine.) If the PURGE operand is not specified, QUIESCE is assumed. If I/O cannot be quiesced, then I/O is halted (see PURGE=HALT).

HALT

indicates that all outstanding requests for input/output operations will not be saved when the STAE exit is taken.

NONE

indicates that input/output processing is allowed to continue normally when the STAE exit is taken.

Notes: If any IBM-supplied access method, except EXCP, is being used, the PURGE=NONE option is recommended. If this is done, all control blocks affected by input/output processing may continue to change during STAE exit routine processing.

If PURGE=NONE is specified and the ABEND was originally scheduled because of an error in input/output processing, an ABEND recursion will develop when an input/output interruption occurs, even if the exit routine is in progress. Thus, it will appear that the exit routine failed when in reality input/output processing was the cause of the failure.

STAE

72.16 ASYNCH=

YES
indicates that asynchronous interrupt processing is allowed to interrupt the processing done by the STAE exit routine. ASYNCH=YES must be coded if:

- Any supervisor services that require asynchronous interruptions to complete their normal processing are going to be requested by the STAE exit routine.
- PURGE=QUIESCE is specified for any access method that requires asynchronous interruptions to complete normal input/output processing.
- PURGE=NONE is specified and the CHECK macro instruction is issued in the STAE exit routine for any access method that requires asynchronous interruptions to complete normal input/output processing.

Note: If ASYNCH=YES is specified and the ABEND was originally scheduled because of an error in asynchronous exit handling, an ABEND recursion will develop when an asynchronous interruption occurs. Thus, it will appear that the exit routine failed when in reality asynchronous exit handling was the cause of the failure.

NO
indicates that asynchronous interrupt processing is not allowed to interrupt the processing done by the STAE exit routine. If the ASYNCH operand is not specified, NO is assumed.

72.17 ISAM Notes: If ISAM is being used and PURGE=HALT is specified or PURGE=QUIESCE is specified but I/O is not restored:

- Only the input/output event on which the purge is done will be posted. Subsequent event control blocks (ECBs) will not be posted.
- The ISAM Check routine will treat purged I/O as normal I/O.
- Part of the data set may be destroyed if the data set is being updated or added to when the failure occurred.

72.18 Control is returned to the instruction following the STAE macro instruction. When control is returned, register 15 contains one of the following return codes:

Hex Code	Meaning
00	Indicates successful completion of creating, overlaying, or canceling a STAE request.
04	Indicates that STAE was unable to obtain storage for the STAE request.
08	Indicates that the user was attempting to cancel or overlay a nonexistent STAE request, or that the user issued an STAE in his STAE exit routine.
0C	Indicates that the exit routine or parameter list address was invalid.
10	Indicates that the user was attempting to cancel or overlay a STAE request of another user.

STAE -- List Form

73.1 The list form of the STAE macro instruction is used to construct a control program parameter list.

73.2 The description of the standard form of the STAE macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the list form only.

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

[symbol]	STAE	[exit address][,PARAM=list address] [,PURGE={QUIESCE, HALT, NONE}] [,ASYNCH={YES, NO}] ,MF=L

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

MF=L
indicates the list form of the STAE macro instruction.

STAE - E Form

STAE -- Execute Form

74.1 A remote control program parameter list is used in, and can be modified by, the execute form of the STAE macro instruction. The control program parameter list can be generated by the list form of the STAE macro instruction. If the user desires to dynamically change the contents of the remote STAE parameter list, he may do so by coding a new exit address and/or a new parameter list address. If exit address or PARAM= is coded, only the associated field in the remote STAE parameter list will be changed. The other field will remain as it was before the current STAE request was made.

74.2 The description of the standard form of the STAE macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the execute form only.

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

```
[symbol]  STAE  [ 0            ] [,OV]
                [ exit address ] [,CT] [,PARAM=list address]

                [,XCTL={YES}]
                [      {NO }]

                [,PURGE={QUIESCE}] [,ASYNCH={YES}]
                [       {HALT   }] [        {NO }]
                [       {NONE   }]

                ,MF=(E,{remote list address})
                       {(1)                }
```

(In the format box, CT and NO are underlined as defaults.)

address
: is any address that is valid in an RX-type instruction, or one of the general registers 2 through 12, previously loaded with the indicated address. The register can be designated symbolically or as an absolute expression, and is always coded within parentheses.

OV
: indicates that the contents of the STAE parameter list will overlay the existing data in the current STAE request.

CT
: indicates that a new STAE request will be created.

MF=(E,{remote list address | (1)})
: indicates the execute form of the STAE macro instruction using a remote parameter list. The address of the remote parameter list can be loaded into register 1, in which case MF=(E,(1)) should be coded.

STATUS -- Change Subtask Status (MVT only)

75.1 The STATUS macro instruction lets the problem programmer change the dispatchability status of one or all of his program's subtasks. One use of the STATUS macro instruction is to restart subtasks that were stopped when an attention exit routine was entered.

75.2 The STATUS macro instruction is used only in an MVT environment. It is ignored when it is issued in MFT.

75.3 The STATUS macro instruction is written as follows:

```
┌─────────┬───────┬──────────────────────────────────────────────────────┐
│[symbol] │STATUS │{START}[,TCB=subtask tcb address]                      │
│         │       │{STOP }                                                │
└─────────┴───────┴──────────────────────────────────────────────────────┘
```

START
: indicates that the STOP/START count in the task control block specified in the TCB operand will be decremented by 1. If the TCB operand is not coded, the STOP/START count is decremented by one in all the subtask task control blocks of the originating task. When the STOP/START count in a TCB reaches 0, the nondispatchability status, established by a previous STATUS macro instruction, is removed.

STOP
: indicates that the STOP/START count in the task control block specified in the TCB operand will be incremented by 1. If the TCB operand is not coded, the STOP/START count is incremented by 1 in the task control blocks for all the subtasks of the originating task. Each task represented by a TCB that has a nonzero STOP/START count is nondispatchable.

TCB= RX-type, (2-12)
: specifies the address of a fullword on a fullword boundary containing the address of the task control block that is to have its STOP/START count adjusted. If a register is designated, the register must contain the address of the task control block. If this operand is not specified, the STOP/START count is adjusted in the task control blocks for all the subtasks of the originating task.

75.4 Control is returned to the instruction following the STATUS macro instruction. When control is returned, register 15 contains one of the following return codes:

Hex Code	Meaning
00	Successful
04	The specified task control block does not belong to a subtask of the originating task. The STATUS macro instruction was ignored.

STIMER -- Set Interval Timer (MFT Without Interval Timer Option)

76.1 When used in an operating system without the interval timer option, the STIMER macro instruction results in an effective NOP instruction. This assures compatibility with an operating system that does include the timer option. The STIMER macro instruction is written as follows:

```
[symbol] | STIMER | {REAL [,timer completion exit address]}
         |        | {TASK [,timer completion exit address]}
         |        | {WAIT                                 }
         |        |
         |        | {,DINTVL=address  }
         |        | {,BINTVL=address  }
         |        | {,TUINTVL=address }
         |        | {,TOD=address     }
```

STIMER -- Set Interval Timer (MFT With Interval Timer Option)

77.1 The STIMER macro instruction causes the control program to set a programmed timer to a specified time interval (less than 24 hours) or to an interval that will expire at a specified time of day. The interval is decremented continuously. An optional timer completion routine is given control after an interruption caused by the interval reaching zero; if no timer completion routine is specified, no indication that the time interval has completed is provided. Only one time interval is in effect at any one time. A second STIMER macro instruction issued before the first time interval has been completely decremented overrides the first interval and exit routine.

77.2 The STIMER macro instruction is written as shown in the following format description. The operands in the shaded area of the format description function in a different manner when used with MFT than when used with MVT. A comparison of the different functions should be made if the macro instruction is coded for upward compatibility. The operands in the nonshaded area can be used with any configuration of the operating system.

```
[symbol]  STIMER   { REAL [,timer completion exit address] }
                   { TASK [,timer completion exit address] }
                   { WAIT                                  }

                   { ,DINTVL=address  }
                   { ,BINTVL=address  }
                   { ,TUINTVL=address }
                   { ,TOD=address     }
```

77.3 REAL
is written as shown. It specifies that the timer interval is to be decremented continuously, and if the TOD operand is coded, the interval will expire at the indicated time of day.

77.4 TASK
is written as shown. It specifies that the timer interval is to be decremented only when the associated task is active.

77.5 WAIT
is written as shown. It specifies that the time interval is to be decremented continuously, and that the associated task is to be placed in the wait condition until the interval is completed.

77.6 timer completion exit address RX-type, (0,2-12)
is the address of the timer completion exit routine to be scheduled to be given control after completion of the specified time interval. The exit routine is given control by means of an interruption of the task that was active when the STIMER macro instruction was issued; the routine must be in main storage when it is required. The contents of the registers when the exit routine is given control are as follows:

Register	Contents
0 - 1	Control program information.
2 - 12	Unpredictable.
13	Address of a control program - provided save area.
14	Return address (to the control program).
15	Address of the exit routine.

The exit routine is responsible for saving and restoring registers.

77.7 DINTVL= RX-type, (1-12)
is the address in main storage of a doubleword on a doubleword boundary containing the time interval. The time interval is presented as unpacked decimal digits of the form:

HHMMSSth, where:

HH is hours (24-hour clock);
MM is minutes;
SS is seconds;
t is tenths of seconds; and
h is hundredths of a second.

77.8 BINTVL= RX-type, (1-12)
is the address in main storage of a fullword on a fullword boundary containing the time interval. The time interval is presented as an unsigned 32-bit binary number; the low-order bit has a value of 0.01 second.

77.9 TUINTVL= RX-type, (1-12)
is the address of a fullword on a fullword boundary containing the time interval. The time interval is presented as an unsigned 32-bit binary number; the low-order bit has a value of one timer unit (26.04166 microseconds).

77.10 TOD= RX-type, (1-12)
is the address of a doubleword on a doubleword boundary containing the time of day at which the interval is to be completed. The time of day is presented as unpacked decimal digits of the form HHMMSSth. If TASK is specified, the time of day is interpreted as though the DINTVL operand had been specified.

Notes: The time interval specified by a STIMER macro instruction has no relation to the time interval specified in an EXEC statement.

If issued by a timer completion exit routine, a STIMER macro instruction acts as a NOP instruction.

The value specified must be a valid positive integer (non-zero).

STIMER -- Set Interval Timer (MVT)

78.1 The STIMER macro instruction causes the control program to set a programmed timer to a specified time interval (less than 24 hours) or to an interval that will expire at a specified time of day. The time interval is associated with the task that was active when the STIMER macro instruction was issued. Only one time interval is associated with a task at any one time; therefore, a second STIMER macro instruction issued for the same task overrides the first time interval.

78.2 The time interval is decremented either continuously or only when the associated task is active. The STIMER macro instruction can be used to request the control program to place the task in the wait condition until the time interval has been completely decremented, and can specify a timer completion exit routine to be given control when the interval reaches zero.

78.3 The STIMER macro instruction is written as follows:

```
[symbol]  STIMER   { REAL [,timer completion exit address] }
                   { TASK [,timer completion exit address] }
                   { WAIT                                  }

                   { ,DINTVL=address  }
                   { ,BINTVL=address  }
                   { ,TUINTVL=address }
                   { ,TOD=address     }
```

78.4 REAL

is written as shown. It specifies that the time interval is to be decremented continuously.

78.5 TASK

is written as shown. It specifies that the time interval is to be decremented only when the associated task is active.

78.6 WAIT

is written as shown. It specifies that the time interval is to be decremented continuously, and that the associated task is to be placed in the wait condition until the interval is completed.

78.7 timer completion exit address — RX-type, (0,2-12)

is the address of the timer completion exit routine to be scheduled to be given control after completion of the specified time interval. If this operand is omitted, no indication of the completion of the time interval is provided. The exit routine has the same dispatching priority as the active task, and is given control when it is the highest priority ready task in the system; the routine must be in main storage when it is required. The contents of the registers when the exit routine is given control are as follows:

Register	Contents
0 - 1	Control program information.
2 - 12	Unpredictable.
13	Address of a control program - provided save area.
14	Return address (to the control program).
15	Address of the exit routine.

The exit routine is responsible for saving and restoring registers.

STIMER

78.8 DINTVL= RX-type, (1-12)

is the address in main storage of a doubleword on a doubleword boundary containing the time interval. The time interval is presented as unpacked decimal digits of the form:
HHMMSSth, where:

HH is hours (24-hour clock)
MM is minutes
SS is seconds
t is tenths of a second
h is hundredths of a second

78.9 BINTVL= RX-type, (1-12)

is the address in main storage of a fullword on a fullword boundary containing the time interval. The time interval is presented as an unsigned 32-bit binary number; the low-order bit has a value of 0.01 second.

78.10 TUINTVL= RX-type, (1-12)

is the address of a fullword on a fullword boundary containing the time interval. The time interval is presented as an unsigned 32-bit binary number; the low-order bit has a value of one timer unit (26.04166 microseconds).

78.11 TOD= RX-type, (1-12)

is the address of a doubleword on a doubleword boundary containing the time of day at which the interval is to be completed. The time of day is presented as unpacked decimal digits of the form HHMMSSth. If TASK is specified, the time of day is interpreted as though the DINTVL operand had been specified.

Notes: The time interval specified by a STIMER macro instruction has no relation to the time interval specified in an EXEC statement.

If issued by a timer completion exit routine, a STIMER macro instruction will be honored. However, the STIMER issued from the exit routine should not specify that same exit routine. If it does specify the same exit routine, an infinite loop may occur.

The value specified must be a valid positive integer (non-zero).

TIME -- Provide Date (MFT Without Timer Option)

79.1 The TIME macro instruction causes the control program to return the date in register 1, as packed decimal digits of the form 00 YY DD DF, where:

YY is the last two digits of the year
DDD is the day of the year
F is a sign character that allows the date to be unpacked and printed directly

79.2 The accuracy of the date information depends upon the accuracy of the corresponding information entered by the operator.

79.3 The TIME macro instruction is written as shown in the format description below. The operands in the shaded area of the format description are used only in an operating system that includes the timer option; they are ignored if coded in an operating system that does not include the timer option.

[symbol]	TIME	[DEC BIN TU MIC,address]

TIME -- Provide Time and Date (MFT With Timer Option, MVT)

80.1 The TIME macro insturction causes the control program to return the date in register 1. For the DEC, BIN, and TU operands, the time of day is returned in register 0. For the MIC,address operand, the time of day is returned in the specified address, and register 0 is set to zero. The time of day and date are only as accurate as the corresponding information entered by the operator.

80.2 DATE

the date is returned in register 1 as packed decimal digits of the form 00 YY DD DF, where:

YY is the last two digits of the year
DDD is the day of the year
F is a sign character that allows the data to be unpacked and printed

80.3 TIME

is the time of day, based on a twenty-four-hour clock, returned in the form designated by the operand shown below. The operand can be omitted, in which case DEC is assumed.

The TIME macro instruction is written as follows:

[symbol]	TIME	⎡DEC⎤ ⎢BIN⎥ ⎢TU⎥ ⎣MIC,address⎦

80.4 DEC

is written as shown. Time of day is returned in register 0 as packed decimal digits of the form:

HHMMSSth, where:

HH is hours (24 hour clock);
MM is minutes;
SS is seconds;
t is tenths of seconds; and
h is hundredths of seconds.

80.5 BIN

is written as shown. Time of day is returned in register 0 as an unsigned 32-bit binary number. The least significant bit is equivalent to one hundredth of a second.

80.6 TU

is written as shown. Time of day is returned in register 0 as an unsigned 32-bit binary number. The least significant bit is equivalent to 26.04166 microseconds (one timer unit).

80.7 MIC

is written as shown.

80.8 address RX-type, (0,2-12)

is the address of an 8-byte area in storage where the time of day is returned as an unsigned binary number with bit 51 equivalent to one microsecond. The MIC,address operand is used only in an MFT or MVT system that has been generated for the IBM System/370.

80.9 If the MIC,address operand is specified, register 15 will contain one of the following return codes when control is returned to the user:

Hex Code	Meaning
00	Successful.
04	Unsuccessful. The specified address is not valid; the date is in register 1, register 0 is zero.

TTIMER -- Test Interval Timer (MFT Without Interval Timer Option)

81.1 When used in an operating system without the interval timer option, the TTIMER macro instruction results in an effective NOP instruction. This assures compatibility with an operating system that does include the timer option. The TTIMER macro instruction is written as follows:

[symbol]	TTIMER	[CANCEL]

TTIMER -- Test Interval Timer (MFT With Interval Timer Option, MVT)

82.1 The TTIMER macro instruction causes the control program to return in register 0 the amount of time remaining in a timer interval previously set by a STIMER macro instruction. The time remaining is returned as an unsigned 32-bit binary number specifying the number of timer units (26 micro-second units) remaining in the interval. If a time interval has not been set, register 0 contains a zero. The TTIMER macro instruction can also be used to cancel the remaining time interval.

82.2 The TTIMER macro instruction is written as follows:

[symbol]	TTIMER	[CANCEL]

CANCEL
: is written as shown. It indicates that the remaining time interval and exit routine, if any, are to be cancelled. If this operand is not designated, the unexpired portion of the time interval remains in effect.

WAIT -- Wait for One or More Events

83.1 The WAIT macro instruction informs the control program that performance of the active task cannot continue until one or more specific events, each represented by a different event control block, have occurred. Bit 0 of each event control block must be set to zero before the event control block is used; the control program takes the following action:

- For each event that has already occurred (each event control block already posted), one is subtracted from the number of events.
- If the number of events is zero by the time the last event control block is checked, control is returned to the instruction following the WAIT macro instruction.
- If the number of events is not zero by the time the last event control block is checked, control is not returned to the issuing program until sufficient event control blocks are posted to bring the number to zero. Control is then returned to the instruction following the WAIT macro instruction.

83.2 A full description of the event control block is presented in the publication IBM System/360 Operating System: System Control Blocks.

83.3 The WAIT macro instruction is written as follows:

[symbol]	WAIT	[number of events,] {ECB=address / ECBLIST=address}

number of events — Sym, Dec Dig, (0,2-12)
maximum is 255. Zero is an effective NOP instruction; one is assumed if the operand is omitted. The number of events must not exceed the number of event control blocks.

ECB= — RX-type, (1-12)
is the address of the event control block representing the single event that must occur before processing can continue. Valid only if the number of events is one or is omitted.

ECBLIST= — RX-type, (1-12)
is the address of a main storage area containing one or more consecutive fullwords on a fullword boundary. Each fullword contains the address of an event control block; the high-order bit in the last word must be set to one to indicate the end of the list. The number of event control blocks must be equal to or more than the specified number of events.

Note: If the program issuing the WAIT has a protection key other than zero, the ECB specified must be in main storage that has the same protection key, except for the communications ECB.

WAITR -- Wait for One or More Events

84.1 The WAITR macro instruction is coded and is executed in exactly the same manner as the WAIT macro instruction.

WTL -- Write to Log

85.1 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 character (C)-type DC statement, and is assembled as a variable-length record.

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

[symbol]	WTL	'message'

message
: is the message to be written to the system log. The message must be enclosed in apostrophes, which will not appear in the log. The message is limited to 126 characters.

WTL -- List Form

86.1 The list form of the WTL macro instruction is used to construct a control program parameter list. The message operand must be provided in the list form of the macro instruction. The description of the standard form of the macro instruction provides the requirements for writing the message.

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

[symbol]	WTL	'message',MF=L

message
is any character string valid in a character (C)-type DC instruction.

MF=L
indicates the list form of the WTL macro instruction.

WTL - E Form

WTL -- Execute Form

87.1 A remote control program parameter list is used in the execute form of the WTL macro instruction. The parameter list can be generated by the list form of the WTL macro instruction. The message cannot be modified in the execute form of the macro instruction.

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

[symbol]	WTL	MF=(E,{control program list address / (1)})

MF=(E,{control program list address / (1)})

indicates the execute form of the macro instruction using a remote control program parameter list. The address of the control program parameter list can be loaded into register 1, in which case MF=(E,(1)) should be coded. If the address is not loaded into register 1, it can be coded as any address that is valid in an RX-type instruction, or one of the general registers 2-12, previously loaded with the address. A register can be designated symbolically or with an absolute expression, and is always coded within parentheses.

WTO -- Write to Operator (Without Multiple Console Support)

88.1 The WTO macro instruction causes a message to be written to the operator's console and/or the system message class data set, depending on the routing and descriptor codes specified.

88.2 The standard form of the WTO macro instruction is written as shown below. The operands in the shaded area of the format description are used in operating systems that include the Multiple Console Support (MCS) option; they are ignored if coded in an operating system that does not include the MCS option, except for routing code 11 which designates a Write-to-Programmer request (WTP) and descriptor codes 1 and 2.

88.3 If a WTO macro instruction is coded with a routing code of 11 in an operating system that does not include the MCS option, this message will go to the system message class data set and will not go to the operator's console. If you want the message to also appear on the operator's console, code the appropriate routing code (as described in Appendix A) in addition to routing code 11. Figure 65 illustrates the type of request resulting from routing and descriptor code requests. Messages that are prefixed by an asterisk indicate a need for operator action. Request types in parentheses result when WTP is not functional.

Routing Code	Descriptor Code	Type of Request
11	None	WTP with no asterisk (WTO with no asterisk)
11	1 or 2	WTP with no asterisk (WTO with asterisk)
11, other	None	WTP and WTO with no asterisk (WTO with no asterisk)
11, other	1 or 2	WTP with no asterisk and WTO with asterisk (WTO with asterisk)
None	1 or 2	WTO with asterisk
None	None	WTO with no asterisk

Figure 65. Routing/descriptor code combinations and resulting actions

Note: The multiple line form of the WTO macro instruction cannot be used to write messages to the system message class data set (Write-to-Programmer messages).

88.4 The operands in the nonshaded area can be coded with any configuration of the operating system.

[symbol]	WTO	{'message' / ('text'[,line type]),...} [,ROUTCDE=(number[,number],...)] [,DESC=(number[,number],...)]

88.5 message
is the message to be written to the operator's console. The message must be enclosed in apostrophes that will not appear on the

console. It can include any character that can be used in a character (C-type) DC instruction, except the New Line control character (punch combination 11-9-5). The maximum message length is 124 characters (bytes) in a system with MVT, and 120 characters in a system with MFT. The message is assembled as a variable-length record.

88.6 ('text'[,line type])
is used to write a multiple-line message to the operator. The message may be up to ten lines long (if more than ten lines are passed by a program, the system will truncate the message at the end of the tenth line). This limit does not include the control line (message IEE932I).

88.7 text
is one line of the multiple-line message to be passed to the operator. A line consists of a character string enclosed in apostrophes (the apostrophes will not appear on the operator's console). Any character valid in a C-type DC instruction may be coded. The maximum number of characters depends on which line type is specified and which version of the operating system (MFT or MVT) is used (see Figure 66).

88.8 line type
is an alphabetic indicator defining 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 will generate a control line (message IEE932I) containing only a message identification number.

L
indicates that the 'text' parameter is a label line. Label lines contain message heading information. If coded, label lines must either immediately follow the control line or be the first line of the multiple-line message if there is no control line. Only 2 label lines may be coded per message.

D
indicates that the 'text' parameter contains the information to be conveyed to the operator by the multiple-line message.

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.

WTO -- Write to Operator (With Multiple Console Support)

89.1 The WTO macro instruction causes a message to be written to one or more operator consoles.

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

```
[symbol]   WTO    {'message'                        }
                  {('text'[,line type]),...         }
                  [,ROUTCDE=(number[,number],...)]
                  [,DESC=(number[,number],...)]
```

89.3 message
is the message to be written to one or more operator consoles. The message must be enclosed in apostrophes that will not appear on the console. It can include any character that can be used in a character (C-type) DC instruction, except the New Line control character (punch combination 11-9-5). The maximum message length is 124 characters (bytes) in a system with MVT, and 120 characters in a system with MFT. The message is assembled as a variable-length record.

89.4 Note: All WTO messages with a descriptor code of 1 or 2 are action messages. An asterisk is printed before the first character of an action message to indicate a need for operator action, but this does not reduce the maximum length of an action message.

89.5 ('text'[,line type])
is used to write a multiple-line message to the operator. The message may be up to ten lines long (if more than ten lines are passed by a program, the system will truncate the message at the end of the tenth line). This limit does not include the control line (message IEE932I).

text
is one line of the multiple-line message to be passed to the operator. A line consists of a character string enclosed in apostrophes (the apostrophes will not appear on the operator's console). Any character valid in a C-type DC instruction may be coded. The maximum number of characters depends on which line type is specified and which version of the operating system (MFT or MVT) is used (see Figure 66).

line type
is an alphabetic indicator defining 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 will generate 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).

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 be the first line of the multiple-line message if there is no control line. Only 2 label lines may be coded per 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.

89.6 ROUTCDE= Dec Dig
specifies the routing codes to be assigned to the message. Number must be a routing code from 1 through 16. Routing codes are defined in Appendix A. If the ROUTCDE operand is omitted but the DESC is specified, routing code 2 is assigned.

89.7 DESC= Dec Dig
specifies the message descriptor code or codes to be assigned to the message. Number must be a descriptor code from 1 through 16. Descriptor codes are defined in Appendix A. If the DESC operand is omitted, no description code is assigned.

89.8 If both the ROUTCDE and DESC parameters are omitted, the routing code specified in the OLDWTOR operand of the system generation SCHEDULR macro instruction is assigned. If the OLDWTOR operand is omitted, no routing code 2 is assigned.

When control is returned, general register 1 contains the identification number (24 bits and right-justified) assigned to the message.

Note: The two operands available to the system programmer are MSGTYP and MCSFLAG. They are discussed in Appendix A.

Line Type	MFT	MVT
C	31 characters	35 characters
L	71 characters	71 characters
D	71 characters	71 characters
DE	71 characters	71 characters
Note: L, D, and DE lines displayed on a 2250 display console will be truncated to 70 characters.		

Figure 66. Maximum 'text' field characters in a multiple-Line WTO message

WTO -- List Form

90.1 The list form of the WTO macro instruction is used to construct a control program parameter list. The message operand must be provided in the list form of the macro instruction. The description of the standard form of the macro instruction provides the requirements for writing the message.

90.2 The format description below indicates the optional and required operands for the list form. The operands in the shaded area of the format description are used with MFT and MVT when those systems include the Multiple Console Support (MCS) option; they are ignored if coded without MCS, except routing code 11 in the ROUTCDE operand which designates a Write-to-Programmer request and descriptor codes 1 and 2. (See the standard form of the WTO macro instruction without MCS for a description of this exception.)

[symbol]	WTO	{('text'[,line type]),...} {'message'} [,ROUTCDE=(number[,number],...)] [,DESC=(number[,number],...)],MF=L

message
    is a character string valid in a character (C-type) DC instruction.

'text'
    is a character string valid in a C-type DC instruction.

line type
    is an alphabetic symbol indicating the type of information contained in the 'text' parameter.

ROUTCDE=
    is one or more decimal digits in the range 1 through 16.

DESC=
    is one or more decimal digits in the range 1 through 16.

MF=L
    indicates the list form of the WTC macro instruction.

Note: Two additional operands available to the system programmer (MSGTYP and MCSFLAG) are discussed in Appendix A.

91.1

WTO - E Form

WTO -- Execute Form

91.1 A remote control program parameter list is used in the execute form of the WTO macro instruction. The parameter list can be generated by the list form of the WTO macro instruction. The message cannot be modified in the execute form of the macro instruction.

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

```
┌─────────┬─────────┬──────────────────────────────────────────────────┐
│[symbol] │  WTO    │ MF=(E,{control program list address})            │
│         │         │       {   (1)                       }            │
└─────────┴─────────┴──────────────────────────────────────────────────┘
```

```
MF(E,{control program list address})
     {     (1)                      }
```

indicates the execute form of the macro instruction using a remote control program parameter list. The address of the control program parameter list can be loaded into register 1, in which case MF=(E,(1)) should be coded. If the address is not loaded into register 1, it can be coded as any address that is valid in an RX-type instruction, or one of the general registers 2-12, previously loaded with the address. A register can be designated symbolically or with an absolute expression, and is always coded within parentheses.

Note: The remote control program parameter list specified must be aligned on a halfword boundary. (The list form of the WTO macro instruction provides this alignment.)

WTOR -- Write to Operator With Reply (Without Multiple Console Support)

92.1 The WTOR macro instruction causes a message requiring a reply to be written to the operator's console, and provides the information required by the control program to return the reply to the issuing program. The program issuing the WTOR should ensure that an acceptable reply is received.

92.2 The standard form of the WTOR macro instruction is written as shown below. The operands in the shaded area of the format description are used in an operating system that includes the Multiple Console Support (MCS) option; they are ignored if coded in an operating system that does not include the Multiple Console Support option, except for routing code 11, which designates a Write-to-Programmer request. If a WTOR message is coded with a routing code of 11 in an operating system that does not include the MCS option, the WTO portion of the message will go to both the system message class data set and the operator's console. The operands in the nonshaded area can be coded with any configuration of the operating system.

[symbol]	WTOR	'message',reply address,length of reply, ecb address [,ROUTCDE=(number[,number],...)] [,DESC=(number[,number],...)]

92.3 message
is the message to be written to the operator's console. The message must be enclosed in apostrophes, which will not appear on the console. It can include any character that can be used in a character (C-type) DC instruction, except the New Line control character (punch combination 11-9-5). The maximum message length is 121 characters (bytes) in a system with MVT, and 117 characters in a system with MFT. The message is assembled as a variable-length record. No requirement exists to pad the message with blanks.

92.4 Note: All WTOR messages are action messages. An asterisk is printed before the first character of an action message to indicate a need for operator action; this does not reduce the maximum length of an action message.

92.5 reply address — A-type, (2-12)
is the address in main storage of the area into which the control program is to place the reply. The reply is left-justified at this address.

92.6 length of reply — Sym, Dec Dig, (2-12)
is the length in bytes, of the reply message. The maximum reply length is 121 bytes.

92.7 ecb address — A-type, (2-12)
is the address of the event control block to be used by the control program to indicate the completion of the reply.

WTOR -- Write to Operator With Reply (With Multiple Console Support)

93.1 The WTOR macro instruction causes a message requiring a reply to be written to one or more operator consoles and the system log, and provides the information required by the control program to return the reply to the issuing program. The program issuing the WTOR should ensure that an acceptable reply is received.

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

```
[symbol]   WTOR    'message',reply address,length of reply,
                   ecb address[,ROUTCDE=(number[,number],...)]
                   [,DESC=(number[,number],...)]
```

93.3 message
is the message to be written to the operator's console. The message must be enclosed in apostrophes, which will not appear on the console. It can include any character that can be used in a character (C-type) DC instruction, except the New Line control character (punch combination 11-9-5). The maximum message length is 121 characters (bytes) in a system with MVT, and 117 characters in a system with MFT. The message is assembled as a variable-length record. No requirement exists to pad the message with blanks.

93.4 Note: All WTOR messages are action messages. An asterisk is printed before the first character of an action message to indicate a need for operator action; this does not reduce the maximum length of an action message.

93.5 reply address — A-type, (2-12)
is the address in main storage of the area into which the control program is to place the reply. The reply is left-justified at this address.

93.6 length of reply — Sym, Dec Dig, (2-12)
is the length, in bytes, of the reply message. The maximum reply length is 121 characters.

93.7 ecb address — A-type, (2-12)
is the address of the event control block to be used by the control program to indicate the completion of the reply.

93.8 ROUTCDE= — Dec Dig
specifies the routing codes to be assigned to the message. Number must be a routing code from 1 through 16. Routing codes are defined in Appendix A. If the ROUTCDE operand is omitted but the DESC operand is specified, routing code 2 is assigned.

93.8 DESC= — Dec Dig
specifies the message descriptor code or codes to be assigned to the message. Number must be a descriptor code from 1 through 16. Descriptor codes are defined in Appendix A. If the DESC operand is omitted, no descriptor code is assigned.

93.9 If both the ROUTCDE and DESC operands are omitted, the routing code specified in the OLDWTOR operand of the system generation SCHEDULR macro instruction is assigned. If the OIDWTCR operand is omitted, routing code 2 is assigned.

93.10 When control is returned, register 1 contains the identification number (24 bits and right-justified) assigned to the message.

Note: The two operands available to the system programmer are MSGTYP and MCSFLAG. They are discussed in Appendix A.

WTOR -- List Form

94.1 The list form of the WTOR macro instruction is used to construct a control program parameter list. The message operand must be provided in the list form.

94.2 The description of the standard form of the WTOR macro instruction provides the requirements for writing the message and the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the list form only. The operands in the shaded area of the format description are used with MFT and MVT when those systems include the Multiple Console Support (MCS) option; they are ignored if coded with MFT or MVT without MCS, except routing code 11 in the ROUTCDE operand which designates a Write-to-Programmer request and descriptor codes 1 and 2. (See the standard form of the WTOR macro instruction without MCS for a description of this exception.)

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

[symbol]	WTOR	'message',[reply address],[length of reply] ,[ecb address] [,ROUTCDE=(number[,number],...)] [,DESC=(number[,number],...)],MF=L

address
is any address that can be written in an A-type address constant.

length
is any absolute expression valid in the assembler language.

message
is a character string valid in a character (C-type) DC instruction.

ROUTCDE=
is one or more decimal digits in the range 1 through 16.

DESC=
is one or more decimal digits in the range 1 through 16.

MF=L
indicates the list form of the WTOR macro instruction.

Note: Two additional operands (MSGTYP and MCSFLAG) available to the system programmer are discussed in Appendix A.

WTOR - E Form

WTOR -- Execute Form

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

95.2 The description of the standard form of the WTOR macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the execute form only. The comma before the first operand is required to indicate the absence of the message operand, which is not allowed in the execute form.

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

```
[symbol]  WTOR   ,[reply address],[length of reply],[ecb address]
                 ,MF=(E,{control program list address})
                        {   (1)                       }
```

address
: is any address that is valid in an RX-type instruction, or one of general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

length
: is any absolute expression that is valid in the assembler language, or one of general registers 2 through 12, previously loaded with the indicated value. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

MF=(E,{control program list address}) / {(1)}
: indicates the execute form of the macro instruction using a remote control program parameter list. The address of the control program parameter list can be coded as described under address, or can be loaded into register 1, in which case MF=(E,(1)) should be coded.

Note: The remote control program parameter list specified must be aligned on a fullword boundary. (The list form of the WTOR macro instruction provides this alignment.)

XCTL -- Pass Control to a Program in Another Load Module

96.1 The XCTL macro instruction causes control to be passed to a specified entry point; the entry point name must be a member name or an alias in a directory of a partitioned data set. (With MVT, the entry point can be an added entry point specified in an IDENTIFY macro instruction.) The load module containing the program is brought into main storage if a usable copy is not available. (Refer to Section I for a discussion of the use of an existing copy of the load module.)

96.2 No return is made to the program issuing the XCTL macro instruction; the responsibility count for the load module containing the XCTL macro instruction is lowered by one. Registers 2 through 14, the program interruption control area, and the program mask must be restored to the conditions that existed when the load module received control before the XCTL macro instruction can be issued. If the specified entry point cannot be located, the task is abnormally terminated.

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

```
[symbol] | XCTL | [(reg1[,reg2])],{EP=symbol                }
         |      |                 {EPLOC=address of name    }
         |      |                 {DE=address of list entry }
         |      |
         |      | [,DCB=dcb address][,HIARCHY=number]
```

96.4 (reg1,[reg2]) — Dec Dig, A-type

is the range of registers from 2 through 12 to be restored from the save area pointed to by register 13. The value of the reg1 operand must be less than the value of the reg2 operand. If the reg2 operand is omitted, only the register specified is loaded; if both operands are omitted, the contents of the registers are not altered.

Note: If a base register is to be restored to its original contents, use the reg1,reg2 operand. Do not change the base register before the XCTL macro instruction is executed.

96.5 EP= — Sym

is the entry point name in the program to be given control.

96.6 EPLOC= — A-type, (2-12)

is the address of the entry point name described above. The name must be padded with blanks to eight bytes, if necessary.

96.7 DE= — A-type, (2-12)

is the address of the name field of a list entry for the entry point name. The list entry is constructed using the BLDL macro instruction. The DCB operand must indicate the same data control block used in the BLDL macro instruction. If the module is indicated as being in the job, step, or task library by the Z byte in the BLDL list entry, the XCTL macro instruction must be either in the same task as the BLDL or in a task with the same chain of task libraries.

96.8 DCB= A-type, (2-12)
is the address of the data control block for the partitioned data set containing the entry point name described above.

If the DCB= operand is omitted or if DCB=0 is specified when the XCTL macro instruction is issued by the job step task, the data sets referenced by either the STEPLIB or JOBLIB DD statement are first searched for the entry point name. If the entry point name is not found, the link library is searched.

If the DCB= operand is omitted or if DCB=0 is specified when the XCTL macro instruction is issued by a subtask, the data set(s) associated with one or more data control blocks referenced by previous ATTACH macro instructions in the subtasking chain are first searched for the entry point name. If the entry point name is not found, the search is continued as if the XCTL macro instruction had been issued by the job step task. The DCB must not be defined in the program issuing the XCTL.

96.9 HIARCHY= Dec Dig
specifies the storage hierarchy (0 or 1) in which the load module is to be loaded when a usable copy is not already available in main storage. If the HIARCHY parameter is missing, loading will take place according to the hierarchy specified at Link Edit time. If HIARCHY is specified, it will override any hierarchy assignments made during linkage editing. The HIARCHY operand is ignored in an operating system that does not have main storage hierarchy support.

96.10 Note: A problem program parameter list may be passed to the called program by loading its address into register 1. The parameter list must begin on a fullword boundary. Each fullword in the list must have the high-order byte set to zeros, except for the last fullword, which must have the high-order bit set to 1.

XCTL -- List Form

97.1 Two parameter lists are used in an XCTL macro instruction: a control program parameter list and an optional problem program parameter list. Only the control program parameter list can be constructed in the list form of the XCTL macro instruction. 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 the XCTL macro instruction.

97.2 The description of the standard form of the XCTL macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the list form only.

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

```
[symbol]   XCTL   ┌EP=symbol                   ┐[,DCB=dcb address]
                  │EPLOC=address of name       │
                  └DE=address of list entry    ┘

                  [,HIARCHY=number],SF=L
```

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

SF=L
    indicates the list form of the XCTL macro instruction.

XCTL - E Form

XCTL -- Execute Form

98.1 Two parameter lists are used in the XCTL macro instruction; 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 the XCTL macro instruction. If only the problem program parameter list is remote, operands that require the control program parameter list cause that list to be constructed in line as part of the macro expansion. If only the control program parameter list is remote, no problem program parameters, including the reg1,reg2 operand, can be specified.

98.2 The description of the standard form of the XCTL macro instruction provides the explanation of the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the execute form only.

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

```
[symbol]   XCTL    [(reg1 [,reg2])][,PARAM=(addresses)[,VL=1]]
                   ┌,EP=symbol                 ┐
                   │,EPLOC=address of name     │ [,DCB=dcb address]
                   └,DE=address of list entry  ┘
                   [,HIARCHY=number]

                   ⎧,MF=(E,{problem program list address})
                   ⎪        {      (1)                   }
                   ⎨,SF=(E,{control program list address})
                   ⎪        {    (15)                    }
                   ⎩,MF=(E,{address}),SF=(E, {address})
                            {  (1)  }        {   (15)}
```

98.4 address
is any address that is valid in an RX-type instruction, or one of general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

98.5 PARAM= RX-type, (2-12)
is one or more address parameters, separated by commas, to be passed to the called program. Each address is expanded to a fullword on a fullword boundary beginning at the address specified in the MF operand. Any parameters specified sequentially overlay the existing addresses in the specified list. This operand can only be used if MF=(E, is specified.

98.6 VL=1
causes the high-order bit of the last address parameter to be set to 1. This operand can only be specified if the PARAM operand is specified, and should be used only if the called program expects a variable number of parameters. If the PARAM operand is specified, but the VL=1 operand is omitted, the high-order bit of the last address parameter is set to 0.

98.7 MF=(E,{problem program list address})
{(1)}
indicates the execute form of the macro instruction using a remote problem program parameter list. Any control program parameters

specified are provided in a control program parameter list expanded in line. The address of the problem program parameter list can be coded as described under "address," or can be loaded into register 1, in which case MF=(E,(1)) should be coded.

98.8 SF=(E,{control program list address}) {(15)}

indicates the execute form of the macro instruction using a remote control program parameter list. No problem program parameters can be specified. The address of the control program parameter list can be coded as described under "address," or can be loaded into register 15, in which case SF=(E,(15)) should be coded.

98.9 MF=(E,{address}),SF=(E,{address}) {(1)} {(15)}

indicates the execute form of the macro instruction using both a remote problem program parameter list and a remote control program parameter list. The addresses of the parameter lists are coded or loaded into registers 1 and 15, as explained above.

APPENDIX A: MESSAGE ROUTING FOR MULTIPLE OPERATOR CONSOLES

ROUTING CODES

A.1 Routing codes provide the mechanism to route WTO and WTOR messages to the locations where they are needed. They indicate the functional area or areas to which a message is to be sent. If no routing code is assigned but a descriptor code is assigned, default is to routing code 2. These codes are not printed or displayed as part of the message text. To use routing codes, the system must have either the MFT or MVT control program and must have the Multiple Console Support (MCS) option included at system generation, except when routing code 11 is used to obtain a Write-to-Programmer message in the message output class.

A.2 Routing codes and their definitions are:

Code	Description
1	MASTER CONSOLE. This routing code is for messages that must be sent to the master console because some action is required by the master console operator, or because the message contains information considered critical to the continued operation of the system. The number of messages with this attribute should be kept to a minimum.
2	MASTER CONSOLE INFORMATIONAL. This routing code is for informational messages to the master console operator. Informational messages usually require no action from the operator. If they do, that action should be at the operator's discretion.
3	TAPE POOL. See routing code 4.
4	DIRECT ACCESS POOL. The tape pool and direct access pool routing codes are for messages that contain instructions for volume handling in the tape and disk areas. Messages about error conditions which occur as a result of the operation of these devices may also be assigned one of these routing codes.
5	TAPE LIBRARY. See routing code 6.
6	DISK LIBRARY. The tape library and disk library routing codes are used for any message that specifies tape library information or disk library information.
7	UNIT RECORD POOL. This routing code is for messages about printers, punches, and card readers. The following classes of information should be sent to this pool: • Types of printer chains or trains required. • Carriage control tapes required. • Types of forms or cards required. • Error conditions on unit record equipment.
8	TELEPROCESSING CONTROL. This routing code is for messages relating to teleprocessing.
9	SYSTEM SECURITY. This routing code is for messages of interest to the system security office (such as password messages).

10 SYSTEM/ERROR MAINTENANCE. This routing code is used for any message indicating system errors or uncorrectable I/O errors, and for any message associated with system maintenance.

11 PROGRAMMER INFORMATION. This routing code is for messages of interest to the programmer. The message is included in the message class for the job and written on the system output device.

Multiple-line messages written using the ('text'[,line type]) parameter of the WTO macro instruction cannot be passed to the programmer using routing code 11.

12 EMULATOR INFORMATION. This routing is for messages issued by an emulator program.

13 USER ROUTING CODE. Available for customer usage.

14 USER ROUTING CODE. Available for customer usage.

15 USER ROUTING CODE. Available for customer usage.

16 RESERVED FOR FUTURE USE.

DESCRIPTOR CODES

A.3 Descriptor codes functionally classify WTO and WTOR messages so that they may be properly presented on all consoles and deleted from display type consoles. Each WTO and WTOR message should contain one descriptor code. If no descriptor code is coded in the WTO or WTOR, no descriptor code is assumed. Descriptor codes 1 through 7 are mutually exclusive, and coding more than one descriptor code in a WTO or WTOR macro instruction makes results unpredictable. These codes are not printed or displayed as part of the message text. To use descriptor codes, the system must have the MFT or MVT control program and must have the Multiple Console Support (MCS) option included at system generation.

A.4 Descriptor codes and their definitions are:

Code	Description
1	SYSTEM FAILURE. This descriptor code is for messages that indicate that a catastrophic error has occurred and another IPL of the system is required.
2	IMMEDIATE ACTION REQUIRED. This descriptor code is for messages that request an immediate operator action (completion of the action is required before a task can proceed). WTO messages with descriptor code 2 must be deleted by a Delete Operator Message (DOM) macro instruction when the operator action has been accomplished, or the operator will have to perform the action to delete the messages. WTOR messages with descriptor code 2 do not require the DOM macro instruction. The message is automatically marked as deleteable upon receipt of the corresponding REPLY command.
3	EVENTUAL ACTION REQUIRED. This descriptor code is for messages requesting operator action where a task does not await completion of the action.
4	SYSTEM STATUS. This descriptor code is for messages that indicate the status of the system, such as system task status or a hardware unit status such as uncorrectable I/O errors.

5 IMMEDIATE COMMAND RESPONSE. This descriptor code is for error and nonerror messages that are written as a direct result of an operator or system command.

6 JOB STATUS. This descriptor code is for messages that indicate the status of a job or job step.

7 APPLICATION PROGRAM/PROCESSOR. This descriptor code is for messages issued by problem programs or by processors executed as problem programs. This descriptor code is the End-of-Step message deletion indicator, and all messages with this code are deleted when the job step in which they were issued is terminated. This does not apply to a terminating TSO task.

8 OUT-OF-LINE MESSAGE. This descriptor code is used for one message or a group of one or more messages that is to be displayed out of line. If the device support cannot print a message out of line, the code will be ignored and the message will be printed in line with other messages.

9 DISPLAY/MONITOR RESPONSE. This descriptor code is used for messages that are written in response to an operator's request for information made by means of the DISPLAY or MONITOR command. Descriptor code 9 also ensures that a control line (message IEE932I) is written for the message. Descriptor code 9 must be specified if a message ID is needed in the control line of a multiple-line message. This allows subsequent message deletion.

10-16 Reserved for future use.

OPERANDS FOR USE BY THE SYSTEM PROGRAMMER

A.5 The WTO and WTOR macro instructions have two special operands, the MSGTYP and MCSFLAG operands. These operands should be used only by the system programmer who is thoroughly familiar with the Multiple Console Support (MCS) Communications Task, since improper use of these operands can impede the entire message routing scheme. These operands set flags to indicate that certain system functions must be performed, or that a certain type of information is being presented by the WTO or WTOR.

A.6 The MSGTYP and MCSFLAG operands may be specified on either the standard or list form of the WTO and WTOR macro instruction. The standard form of the WTO macro instruction is shown below.

[symbol]	WTO	{('text'[,line type]) {'message' [,DESC=(number)] [,ROUTCDE=(number)] [,MSGTYP={N | Y | JOBNAMES | STATUS | ACTIVE}] [,MCSFLAG=(name[,name],...)]

A.7 MSGTYP=JOBNAMES or MSGTYP=STATUS
specifies that the message is to be routed to the console which issued the DISPLAY JOBNAMES or DISPLAY STATUS command, respectively. When the message type is identified by the operating system, the message will be routed to only those consoles that had requested the information. Omission of the MSGTYP parameter causes the message to be routed as specified in the ROUTCDE parameter.

A.8 MSGTYP=ACTIVE

specifies that the multiple-line message is in response to a MONITOR A (MN A) command and should be routed to the console that issued the command.

A.9 MSGTYP=Y or MSGTYP=N

specifies that two bytes are to be reserved in the WTO or WTOR macro expansion so that flags can be set to describe what MSGTYP functions are desired (see Figure 67). Y specifies that two bytes of zeros are to be included in the macro expansion at displacement WTO + 4 + the total length of the message text, descriptor code, and routing code fields. N, or omission of the MSGTYP parameter, specifies that the two bytes are not needed, and that the message is to be routed as specified in the ROUTCDE parameter. If an invalid MSGTYP value is encountered, a value of N is assumed, and a diagnostic message is produced (severity code of 8).

Bit	Meaning
0	DISPLAY JOBNAMES
1	DISPLAY STATUS
2-15	Reserved for future system use. Must be zeros.

Figure 67. Bit definitions for MSGTYP=Y

When MSGTYP=Y, the issuer of the WTO or WTOR macro instruction that contains the MSGTYP information must set the appropriate message identifier bit in the MSGTYP field of the macro expansion. Prior to executing the WTO or WTOR SVC (SVC 35), he must also set byte 0 of the MCSFLAG field in the macro expansion to a value of X'10'. This value indicates that the MSGTYP field is to be used for the message routing criteria. When the message type is identified by the system, the message will be routed to all consoles that had requested that particular type of information. Routing codes, if present, will be ignored.

A.10 MCSFLAG

specifies that the macro expansion should set bits in the MCSFLAG field as indicated by each name coded. Names and their corresponding bit settings are shown in Figure 68.

A.11 ROUTCDE, DESC, and MSGTYP parameter combinations are shown in Figure 69. Coding of any one of the four keyword parameters (ROUTCDE, DESC, MSGTYP, MCSFLAG) causes a new format WTO or WTOR to be generated.

Name	Bit	Meaning
----	0	Invalid entry.
REG0	1	Message is to be queued to the console whose source ID is passed in Register 0.
RESP	2	The WTO is an immediate command response.
----	3	Invalid entry.
REPLY	4	The WTO macro instruction is a reply to a WTOR macro instruction.
BRDCST	5	Message should be broadcast to all active consoles.
HRDCPY	6	Message queued for hard copy only. This operand is invalid with the multiple-line form of WTO.
QREG0	7	Message is to be queued unconditionally to the console whose source ID is passed in Register 0.
NOTIME	8	Time is not appended to the message. This operand is invalid with the multiple-line form of WTO.
----	9-12	Invalid entry.
NOCPY	13	If the WTO or WTOR macro instruction is issued by a program in the supervisor state, the message is not queued for hard copy. Otherwise, this parameter is ignored.
----	14-15	Invalid entry.

Note: Invalid specifications are ignored and produce an appropriate error message.

Figure 68. MCSFLAG parameters

No.	Parameter Coded: ROUTCDE	Parameter Coded: DESC	Parameter Coded: MSGTYP	Parameter Coded: MCSFLAG	Expansion Generates: ROUTCDE	Expansion Generates: DESC	Expansion Generates: MSGTYP	Expansion Generates: MCSFLAG
1	Specified	Specified	Y	Optional	Codes Specified	Codes Specified	Zeros	As Specified#
2	Specified	Specified	N	Optional	Codes Specified	Codes Specified	Field Omitted	As Specified#
3	Specified	Specified	JOBNAMES	Optional	Codes Specified	Codes Specified	X'8000'	As Specified#
4	Specified	Specified	STATUS	Optional	Codes Specified	Codes Specified	X'4000'	As Specified#
5	Specified	Specified	Omitted	Optional	Codes Specified	Codes Specified	Field Omitted	As Specified#
6	Specified	Omitted	Y	Optional	Codes Specified	Zeros	Zeros	As Specified#
7	Specified	Omitted	N	Optional	Codes Specified	Zeros	Field Omitted	As Specified#
8	Specified	Omitted	JOBNAMES	Optional	Codes Specified	Zeros	X'8000'	As Specified#
9	Specified	Omitted	STATUS	Optional	Codes Specified	Zeros	X'4000'	As Specified#
10	Specified	Omitted	Omitted	Optional	Codes Specified	Zeros	Field Omitted	As Specified#
11	Omitted	Specified	Y	Omitted*	Routing Code 2	Codes Specified	Zeros	X'8000'
12	Omitted	Specified	N	Omitted*	Routing Code 2	Codes Specified	Field Omitted	X'8000'
13	Omitted	Specified	JOBNAMES	Omitted*	Routing Code 2	Codes Specified	X'8000'	X'8000'
14	Omitted	Specified	STATUS	Omitted*	Routing Code 2	Codes Specified	X'4000'	X'8000'
15	Omitted	Specified	Omitted	Omitted*	Routing Code 2	Codes Specified	Field Omitted	X'8000'
16	Omitted	Specified	Y	REG0/QREG0	Zeros	Codes Specified	Zeros	As Specified#
17	Omitted	Specified	N	REG0/QREG0	Zeros	Codes Specified	Field Omitted	As Specified#
18	Omitted	Specified	JOBNAMES	REG0/QREG0	Zeros	Codes Specified	X'8000'	As Specified#
19	Omitted	Specified	STATUS	REG0/QREG0	Zeros	Codes Specified	X'4000'	As Specified#
20	Omitted	Specified	Omitted	REG0/QREG0	Zeros	Codes Specified	Field Omitted	As Specified#
21	Omitted	Omitted	Y	Omitted*	Routing Code 2	Zeros	Zeros	X'8000'
22	Omitted	Omitted	N	Omitted*	Routing Code 2	Zeros	Field Omitted	X'8000'
23	Omitted	Omitted	JOBNAMES	Omitted*	Routing Code 2	Zeros	X'8000'	X'8000'
24	Omitted	Omitted	STATUS	Omitted*	Routing Code 2	Zeros	X'4000'	X'8000'
25	Omitted	Omitted	Omitted	Omitted*	Field Omitted	Field Omitted	Field Omitted	Zeros
26	Omitted	Omitted	Y	REG0/QREG0	Zeros	Zeros	Zeros	As Specified#
27	Omitted	Omitted	N	REG0/QREG0	Zeros	Zeros	Field Omitted	As Specified#
28	Omitted	Omitted	JOBNAMES	REG0/QREG0	Zeros	Zeros	X'8000'	As Specified#
29	Omitted	Omitted	STATUS	REG0/QREG0	Zeros	Zeros	X'4000'	As Specified#
30	Omitted	Omitted	Omitted	REG0/QREG0	Zeros	Zeros	Field Omitted	As Specified#

* If an MCSFLAG other than REG0 or QREG0 is specified, the expansion generates the same fields except that the MCSFLAG field contains the MCSFLAG specified and the high-order bit set to 1.

\# High order bit set to 1 to indicate a new format macro expansion (routing code and descriptor code fields exist).

Figure 69. ROUTCDE, DESC, and MSCTYP combinations

APPENDIX B: SUMMARY OF OPERANDS

B.1 Figure 70 indicates how each operand may be coded in the standard and, where applicable, in the list and execute forms of each macro instruction. For example, in ATTACH macro instruction the DCB operand may be coded in the standard (S) form using registers 2-12 or as an A-type address constant, in the list (L) form as an A-type address constant, and in the execute (E) form using registers 2-12 or as an RX-type address constant. Only the indicated methods of coding should be used.

Abbreviations Used in Figure 70

Abbreviation		Meaning
Sym		Any symbol valid in the Assembler Language.
Dec Dig		Any decimal digits, up to the value indicated in the associated macro instruction description. If both SYM and DEC DIG are checked, an absolute expression is also allowed.
Register		A general register, always coded within parentheses, as follows:
	(2-12) -	one of the general registers 2 through 12, previously loaded with the right-adjusted value or address indicated in the macro instruction description. The unused high-order bits must be set to zero. The register may be designated symbolically or with an absolute expression.
	(1) -	general register 1, previously loaded as indicated above. The register can be designated only as (1).
	(0) -	general register 0, previously loaded as indicated above. The register can be designated only as (0).
RX-type		Any address that is valid in an RX-type instruction (for example, LA) may be designated.
A-type		Any address that may be written in an A-type address constant may be designated.

Macro Instruction	Operands	Written As: Sym	Dec Dig	Register: (2-12)	(1)	(0)	RX type	A-type
ABEND	completion code	S	S	S	S			
	DUMP	written as shown						
	STEP	written as shown						
ATTACH	ASYNCH=	YES or NO						
	DCB=			S E			E	S L
	DE=			S E			E	S L
	DPMOD=	S L E	S L E	S E				
	ECB=			S E			E	S L
	EP=	S L E						
	EPLOC=			S E			E	S L

Figure 70 (part 1 of 5). Summary of operands

Macro Instruction	Operands	Written as						
				Register				
		Sym	Dec Dig	(2-12)	(1)	(0)	RX type	A-type
	ETXR=			S E			E	S L
	GSPL=			S E			E	S L
	GSPV=	S L E	S L E	S E				
	HIARCHY=		S L E					
	LPMOD=	S L E	S L E	S E				
	PARAM=			S E			E	S
	PURGE=	QUIESCE, HALT, or NONE						
	SHSPL=			S E			E	S L
	SHSPV=	S L E	S L E	S E				
	STAI=			S E			E	S L
	SZERO=	YES or NO						
	TASKLIB=			S E			E	S L
	VL=1	written as shown						
CALL	entry point name	S E						
	address parameters			S E			E	S L
	VL	written as shown						
	ID=	S E	S E					
CHAP	priority change value	S	S	S		S		
	tcb location address			S	S		S	
CHKPT	dcb address			S E			E	S L
	checkid address			S E			E	S L
	checkid length	S L E	S L E	S E				
	CANCEL	written as shown						
DELETE	DE=			S		S	S	
	EP=	S						
	EPLOC=			S		S	S	
DEQ	qname address			S E			E	S L
	rname address			S E			E	S L
	rname length	S L E	S L E	S E				
	STEP or SYSTEM	written as shown						
	RET=HAVE	written as shown						
DETACH	tcb location address	S		S	S		S	
	STAE=	YES or NO						
DOM	MSG=			S	S			
	MSGLIST=	S		S	S		S	

Figure 70 (part 2 of 5). Summary of operands

Macro Instruction	Operands	Written as						
				Register				
		Sym	Dec Dig	(2-12)	(1)	(0)	RX type	A-type
DXR	reg1	S	S					
	reg2	S	S					
ENQ	qname address			S E			E	S L
	rname address			S E			E	S L
	E or S	written as shown						
	rname length	S L E	S L E	S E				
	STEP or SYSTEM	written as shown						
	RET=	TEST, USE or HAVE						
EXTRACT	answer area address			S E			E	S L
	tcb location address			S E			E	S L
	FIELDS=	refer to macro description						
FREEMAIN	E,L,R, or V	written as shown						
	A=(with E,L, or V)			S E			E	S L
	A=(with R)			S	S		S	
	LA=			S E			E	S L
	LV=(with E)	S L E	S L E	S E				
	LV=(with R)	S	S	S		S		
	SP=(with E,L, or V)	S L E	S L E	S E				
	SP=(with R)	S	S	S		S		
GETMAIN	code	refer to macro description						
	A=			SE			E	S L
	HIARCHY=		S L E					
	LA=			S E			E	S L
	LV=(with E)	S L E	S L E	S E				
	LV=(with R)	S	S	S		S		
	SP=(with E,L, or V)	S L E	S L E	S E				
	SP=(with R)	S	S	S		S		
IDENTIFY	ENTRY=			S	S		S	
	EP=	S						
	EPLOC=			S		S	S	
LINK	DCB=			S E			E	S L
	DE=			S E			E	S L
	EP=	S L E						
	EPLOC=			S E			E	S L

Figure 70 (part 3 of 5). Summary of operands

Macro Instruction	Operands	Written as						
				Register				
		Sym	Dec Dig	(2-12)	(1)	(0)	RX type	A-type
	HIARCHY=		S L E					
	ID=	S E	S E					
	PARAM=			S E			E	S
	VL=1	written as shown						
LOAD	DCB=			S	S		S	
	DE=			S		S	S	
	EP=	S						
	EPLOC=			S		S	S	
	HIARCHY=		S					
POST	ecb address			S	S		S	
	completion code	S	S	S		S		
RETURN	(reg 1, reg 2)		S					
	T	written as shown						
	RC=	S	S	or (15)				
SAVE	(reg 1, reg2)		S					
	T	written as shown						
	identifier name	character string or*						
SEGLD	external segment name	S						
SEGWT	external segment name	S						
SNAP	DCB=			S E			E	S L
	ID=	S L E	S L E	S E				
	LIST=			S E			E	S L
	PDATA	refer to macro description						
	SDATA	refer to macro description						
	STORAGE			S E			E	S L
	TCB=			S E			E	S
SPIE	interruption exit address			S E			E	S L
	interruptions		S L E					
STAE	exit address			S E			E	S L
	OV or CT	written as shown						
	PARAM=			S E			E	S L
	ASYNCH=	YES or NO						
	PURGE=	QUIESCE, HALT, or NONE						
	XCTL=	YES or NO						

Figure 70 (part 4 of 5). Summary of operands

Macro Instruction	Operands	Written as						
				Register				
		Sym	Dec Dig	(2-12)	(1)	(0)	RX type	A-type
STATUS	STOP or START	written as shown						
	TCB=			S			S	
STIMER	REAL, TASK or WAIT	written as shown						
	timer completion exit addr			S		S	S	
	BINTVL=			S	S		S	
	DINTVL=			S	S		S	
	TOD=			S	S		S	
	TUINTVL=			S	S		S	
TIME	DEC or BIN or TU	written as shown						
	MIC	written as shown						
	address			S		S	S	
TTIMER	CANCEL	written as shown						
WAIT WAITR	number of events	S	S	S		S		
	ECB=			S	S		S	
	ECBLIST=			S	S		S	
WTL	message	any message within apostrophes						
WTO	message	any message within apostrophes						
	ROUTCDE=		S L					
	DESC=		S L					
WTOR	message	any message within apostrophes						
	reply address			S E			E	S L
	length of reply	S L E	S L E	S E				
	ecb address			S E			E	S L
	ROUTCDE=		S L					
	DESC=		S L					
XCTL	(reg 1, reg 2)		S E				E	S
	DCB=			S E			E	S L
	DE=			S E			E	S L
	EP=	S L E						
	EPLOC=			S E			E	S L
	HIARCHY=		S L E					
	PARAM=			E			E	

Figure 70 (part 5 of 5). Summary of operands

To help you find the information quickly and easily, the entries in this index refer to paragraph numbers rather than to page numbers. Each paragraph number is composed of two parts separated by a point. The first part denotes the chapter number; the second part denotes the sequential position of the paragraph within the chapter. For example, the index entry "writing to the hard copy log 5.80-84" indicates that the information can be found in paragraphs 80 through 84 of chapter 5. For your convenience, the upper left corner of every left-hand page in this book contains the number of the first paragraph on that page.

Indexes to Systems Reference Library publications are consolidated in IBM System/360 Operating System: Systems Reference Library Master Index, GC28-6644. For additional information about any subject listed below, refer to other publications listed for the same subject in the Master Index.

ABEND macro instruction 5.156-192
- abnormal condition handling by 5.156-198
- coding in MFT without subtasking 10.1
- coding in MVT and MFT with subtasking 11.1
- completion code 5.163,10.3,11.4
- interception
  - by STAE 5.166-183
  - by STAI 5.184-185
  - obtaining a dump 5.186-194,10.4,11.5
  - STEP operand 5.161,11.6

abnormal condition 5.156-192
- abnormal termination routine 5.157
- attempting error recovery from 5.166-183
- detection of 5.157
- handling 5.156-198
  - by ABEND 5.159-161

abnormal termination
- interception of 5.166-185
- from program interruption 5.173
- restart after 22.1
- routine 5.157
- of subtask 5.184-185
- of task 5.166-183

additional entry points 5.2-6
- in ATTACH 12.1
- in IDENTIFY 54.1
- in LINK 55.1
- in LOAD 58.2
- in XCTL 96.1

address parameters 2.34

ATTACH macro instruction
- coding in MFT with subtasking 13.1
- coding in MFT without subtasking 12.1
- coding in MVT 14.1
- with DETACH 30.1,31.1
- ECB operand 2.119,4.5-9,5.163,12.10, 13.14,14.15
- ETXR operand 2.119-120,4.5-9,5.163, 12.11,13.15,14.16
- with IDENTIFY 54.1
- STAI operand 5.184,14.23
- STAI retry routine 5.183-185
- SZERO operand 14.22

base register
- initial 2.5
- permanent 2.15

BINTVL (binary interval operand) 5.55

branching table
- example of 2.52
- use when passing control with return 2.52

CALL macro instruction 2.44-49
- coding 17.1
- creating parameter list for LINK, ATTACH, and XCTL 18.1
- passing control using 2.46-49
- results of expansion 2.49

calling program, definition of 2.2

calling sequence identifier 5.8

canceling the current STAE request 5.171,72.3,72.9

CANCEL operand
- (see also timing services)
- in CHKPT 22.15
- in TTIMER 5.54,82.2

CHAP macro instruction 3.20-31,21.1
- (see also priority)

characteristics, load module 2.25-27

checkpoint and restart 7.1

checkpoint data sets 22.4
- defining 22.6

CHKPT macro instruction 22.1
- CANCEL operand 22.15
- return codes 22.16

CLASS parameter of JOB statement with MFT 3.24

coding aids 8.6

command scheduler communications parameter list address 5.44,39.5

completion code
- (see also return code)
- in ABEND 5.163,10.3,11.4
- in ATTACH 14.27
- in event control block (ECB) 12.10,13.14,14.15

in POST 59.2
in task control block (TCB) 5.43,11.4
COND operand
in EXEC statement 2.61,5.158
in JOB statement 2.61,5.158
conditional requests
from DEQ 5.25-31,26.8
from ENQ 5.25-31,35.13
from GETMAIN 6.14-16,46.7
configurations of the operating system 1.3-4
control program options 8.5
core image dump 5.196-198,65.1,66.1
core storage (IBM 2361 Core Storage)
(see main storage hierarchy support)

DCB operand
in ATTACH 2.74,12.7,13.11,14.10
in LINK 2.74,55.7
in LOAD 2.74,58.7
in XCTL 2.74,96.8
DD statement, SYSABEND or SYSUDUMP 10.4,11.5
DE operand
in ATTACH 2.74,12.6,13.10,14.9
in LINK 2.74,55.6
in LOAD 2.74,58.6
in XCTL 2.74,96.7
DELETE macro instruction 2.115,25.1
DEQ macro instruction 26.1
proper use of 5.25-39
using the list and execute forms 6.42
DESC operand 5.71
in WTO 88.3,5.71,89.7
in WTOR 93.8
descriptor codes A.3-4
causing an * in the message 5.71,88.3
designing programs, requirements for 2.1-129
DETACH macro instruction 4.5,4.9,5.163
coding in MFT 30.1
coding in MVT 31.1
DINTVL operand 5.55
in MFT 77.7
in MVT 78.8
dispatching priority 3.10-31
(see also priority)
available in task control block 5.43
changing 3.21,21.1
computing 3.12,3.18
definition of 3.10
DPRTY parameter of EXEC statement 3.12-13
of partitions 3.27
specifying 13.20,14.12
disposing of the message to the operator (with MCS) 5.71
DOM macro instruction 5.90,33.1
DPMOD operand 3.18
in MFT 13.19
in MVT 14.12
DUMP 5.186-198
ABEND 5.188-194,10.4,11.5
core image 5.196-198
indicative 5.195,10.4
requirements 5.186-198
SNAP 5.188-194
in MFT 65.1
in MVT 66.1
DUMP operand in ABEND 5.188,10.4,11.
DXR macro instruction 5.125-139,34.1
dynamic structure 2.22,2.27

ECB (see event control block)
ECB operand
with ATTACH 12.10,13.14,14.15
effect on task termination 5.161
with POST 59.1
with WAIT 83.3
element type (E) explicit request for main storage 6.11,46.4,47.4
end-of-task exit routine 5.45
in MFT without subtasking 12.11
in MFT with subtasking 13.15
in MVT 14.16
ENQ macro instruction
coding 35.1
control program processing of 5.17-21
controlling load module use 2.114
exclusive control 5.16,35.8
proper use of 5.22-39
requesting control of a resource 5.11-16,35.1
restriction on qname parameter 5.14,35.6
shared control 5.16-39,35.9
testing for simultaneous resource use 5.11
unconditional request 35.13
entry point identifier
defined 5.7
specified in SAVE macro instruction 2.10,5.7,61.5
specified in GTRACE macro instruction 50.2
entry points
added
in ATTACH 12.1,13.1,14.1
by IDENTIFY 5.3-4,54.1
in LINK 55.1
in LOAD 58.2
in XCTL 96.1
restrictions for additional 5.5
EP operand 2.74-82
in ATTACH 12.4,13.8,14.7
in DELETE 25.2
in IDENTIFY 54.3
in LINK 55.4
in LOAD 58.4
in XCTL 96.5
EPLOC operand 2.74-82
in ATTACH 12.5,13.9,14.8
in DELETE 25.2
in IDENTIFY 54.3
in LINK 55.5
in LOAD 58.5
in XCTL 96.6
ETXR operand

in ATTACH macro
 instruction 12.11,13.15,14.16
use in MFT without subtasking 2.119-120
use in MVT and MFT with
 subtasking 4.5-6,5.163
use in termination 5.163
event control block (ECB)
in ATTACH 12.10,13.14,14.15
creation of 4.13
diagram of 4.11
in POST 59.1
reusing 4.13
in WAIT 83.1
use with ATTACH, POST, and WAIT 4.10-14
EXEC statement, PARM field 2.21
execute form of macro
 instructions 16.40-42,9.3-4
execution, selection of job steps for 6.2
exit routines
end-of-task exit routine
 (ETXR) 12.11,13.15,14.16
program interruption exit routine 69.1
specifying a task abnormal exit
 routine 72.1
task abnormal exit routine 72.1
timer completion exit routine 77.6,78.7
explicit requests
for main storage 6.6-32
for a resource 5.16-21
extended-precision floating-point
 simulation 5.125-155
EXTRACT macro instruction
coding in MFT without subtasking 38.1
coding in MVT and MFT with
 subtasking 39.1
determining current dispatching
 priority 3.20,5.43
determining initial dispatching
 priority 5.43
determining limit priority 3.20
requires an answer area 5.46,39.5
used to obtain information from the task
 control block 5.43
using FIELDS=ALL 5.46
warning for using task control
 block 4.5,4.9

FIELDS operand (see EXTRACT)
flag, save area 2.57
FREEMAIN macro instruction
coding in MFT 42.1
coding in MVT 43.1
releasing subpools 6.32
restriction regarding subpool 0 6.24
returning control of main
 storage 6.7,6.32

GETMAIN macro instruction
coding in MFT 46.1
coding in MVT 47.1
creating subpools 6.29
explicit request for main
 storage 6.6-32
example 6.15
specifying length of main
 storage 6.10-13
types of 6.9-14
GSPL operand of ATTACH 3.7,6.30,14.19
GSPV operand of ATTACH 3.7,6.30,14.18
generalized trace facility (GTF) 5.96-99
GTRACE macro instruction 5.96-99,50.1
return codes 50.3

halting I/O
in ATTACH 14.24
in STAE 72.15
hard copy log 5.80-84
HIARCHY operand 6.54-55
in ATTACH 12.13,13.17,14.17
in GETMAIN 46.12,47.14
in LINK 55.11
in LOAD 58.8
in XCTL 96.9
hierarchies, main storage 6.51-57
examples using
hierarchy 0 6.15
hierarchy 1 6.23

IDENTIFY macro instruction 54.1
adding entry points 5.3-4
restrictions 5.3
return codes 54.4
identify option 5.2
implicit requests for main storage 6.33-47
ATTACH 6.33,6.45
LINK 6.33,6.45
LOAD 6.33,6.45
OPEN 6.33
XCTL 6.33,6.45
imprecise interruptions 5.116-122
indicative dump (MFT) 5.195
instruction length code (ILC) 5.116
interlock situation 5.32-39
interruptions 5.100-122
(see also program interruption
 processing)
imprecise 5.116-122
precise 5.116-122
interval timing 5.54-64

job class 6.2
job library 2.65-70,2.106
job pack area 2.72-104
job priority
effect on execution 6.3
specifying 3.12-31
job step termination 5.164,11.6

library
definition of 2.65
job 2.65-70,2.106
link 2.66-106
private 2.69
step 2.67-86

limit priority 3.12
(see also priority)
link library 2.66-106
LINK macro instruction 2.98-106
coding 55.1
difference from CALL macro instruction 2.101
implicit request for main storage 6.33,6.45
responsibility count 2.103
similarity to CALL macro instruction 2.100
use in passing control with return 2.98
use with BLDL 2.105
use with the job library 2.104
use with the link library 2.104
use with a private library 2.104
use with a step library 2.104
link pack area (MVT)
contents of 6.49
placing modules in 6.36
searching 2.73
linkage conventions 2.4-20
linkage registers 2.16-20
entry point register 2.20
parameter registers 2.17
return address register 2.19
save area register 2.18
list form of macro instructions 6.40-42,9.2-4
list, parameter list creation by CALL 18.1
list type (L) explicit request for main storage 6.12
LOAD macro instruction 2.93-97,58.1
load module
(see also dynamic structure; overlay structure, planned; simple structure)
attributes 2.89
characteristics 2.28
copy
finding a usable 2.73-86
using an existing 2.87-92
execution
parallel 2.28
serial 2.28-29
management 6.34-47
nonreusable 2.89
temporarily 6.47
reenterable 6.35-38
serially reusable 2.88
structures 2.22-29
log
hard copy 5.80-84
system 5.85-88
WTL 5.87-88
LPMOD operand in ATTACH 3.18,13.18,14.11

machine-check handler 6.37
macro definition listing 8.3
macro instructions defined elsewhere 8.13-16
main storage
blocks
assignment 6.9,6.20
size 6.22
control 6.20-50
efficient use of 6.5
example of assignment 6.23
fragmentation 6.44-47
hierarchies 6.51-57
management 6.1-57
(see also GETMAIN; FREEMAIN; subpool)
release 6.48-50
requests
conditional 6.14-16
control program 6.4
explicit, via GETMAIN 6.6,6.15,6.20
implicit 6.4,6.33
unconditional 6.14-16
reuse 6.49
main storage hierarchy support 6.51-57
hierarchies 6.51
overrun 6.57
use with Model 50 6.57
masking program interruptions with SPIE 69.1
master console operator answering a WTOR 5.75
message deletion 5.89-90
message identifier 5.73,89.9
message output class, specified by MSGCLASS parameter 5.78
messages
(see also writing to the operator; WTO; WTOR)
to the operator 5.65-75
to the programmer 5.76-79,88.2
Model 65 interruptions 5.116-118
Model 67 interruptions 5.116-118
Model 75 interruptions 5.116-118
Model 85 interruptions 5.116-118
Model 91 interruptions 5.116-122
during decimal simulation 5.123-124
Model 195 interruptions 5.116-122
MODIFY command 5.91-5.94
MSGCLASS parameter of the JOB statement 5.78
multiple console support (MCS)
(see descriptor codes; hard copy log; message deletion; routing codes; system log)

new line control character, restriction with WTO 5.66
nonreenterable load modules 6.43-47
nonreusable load modules 2.89,2.115-116

obtaining information from the task control block 5.40-46
old program status word (OPSW) 5.111
operator communications 5.91-5.95
options, control program 8.5
originating task, definition of 3.5
OV operand of STAE 5.171
overlap of task execution 3.4
overlay of a STAE request 5.171
overlay structure, planned

advantages of 6.46
definition of 2.26
passing control in 2.62
overrun, with main storage hierarchy support 6.57

pack areas (see job pack area; link pack area)
parallel execution of a job step, definition of 3.4
parameter list
with CALL 2.46-49,18.1
handling of 2.38-39
inline 2.45-49
with LINK 2.104,55.8
from PARM field 2.38
with XCTL 2.126,96.10
parameters (see parameter list; linkage registers)
PARM field 2.21
partitions (MFT) 6.2
passing control
(see also ATTACH; LINK; XCTL)
in a dynamic structure 2.63,2.98-129
loading the module 2.64-129
with return 2.36-49
without return 2.31-35
in a planned overlay structure 2.62
in a simple structure 2.30-61
with return 2.40-60
without return 2.31-39
PICA (program interruption control area) 5.103-108,69.8
PIE (program interruption element) 5.107-108
planned overlay structure (see overlay structure, planned)
POINT macro instruction, in a reenterable load module 6.39
POST macro instruction 4.10-13,59.1
precise interruptions 5.116-122
priority
assigning 3.10-22,13.19,14.11
changing 3.21-22,21.1
dispatching 3.10-22,5.43
initial dispatching 3.18
limit 3.10,5.43
of partitions 3.25-27
subtask 3.18-122
task 3.10-17
private library
definition of 2.69
searching 2.69-86
program exceptions 5.100
(see also program interruption processing)
program interruption control area (PICA) 5.103-108,69.8
program interruption element (PIE) 5.107-108
program interruption processing 5.100-122
imprecise interruptions 5.116-122
precise interruptions 5.116-122
standard control program exit routine 5.101
user exit routine 5.101-115
for imprecise interruptions 5.122
register contents when control gained 5.109
program management 2.1-129
program management services 5.1-198
(see also abnormal conditions; additional entry points; calling sequence identifiers; deleting messages; dump; entry point identifier; obtaining information from the task control block; processing program interruptions; serially reusable resources; timing services; writing to the hard copy log; writing to the operator; writing to the system log)
protection, of serially reusable resources 5.9-39

QEDIT 5.93-5.94
qname operand of ENQ, restriction on 5.14,35.6

read-only load module (see reenterable load module)
REAL parameter of STIMER 5.57-58,77.3,78.4
reducing main storage required for a job step 6.20-32
reenterable load modules 2.118,6.35-38
in MFT with subtasking 2.90
in MVT 2.88-89
reenterable macro instructions 6.39-42
refreshable load module 6.38
regions (MVT)
controlling 6.3
extending by rollout/rollin 6.3
specifying size on EXEC statement 6.3
specifying size on JOB statement 6.3
register type (R) explicit request for main storage 6.10,46.5,47.6
registers
(see also base register; linkage registers; reenterable macro instructions)
specifying 8.9
releasing main storage 6.48-50
(see also DEQ; FREEMAIN)
reply, (see WTOR)
resident reenterable module area 2.73,2.92
resource
conditionally requesting, via ENQ 5.25,35.13
control 5.13-39
duplicate request for, definition of 5.23
releasing control of with DEQ 5.24-39,26.1
request for, causing interlock 5.32-39
serially reusable 5.9-12
unconditionally requesting, via ENQ 5.25-31,35.13
responsibility count

ensuring that the proper one is lowered 2.128-129
lowering it via the control program 2.129
lowering it via DELETE 2.129,25.1
with release of main storage 6.48
restart
automatic 22.1
deferred 22.3
RET operand 35.13
RET=CHNG 5.28
RET=HAVE 5.29
RET=TEST 5.26
RET=USE 5.27
return code 2.51-52
and ATTACH 3.6,14.27
from BLDL 2.83
with branching table 2.52
and COND operand 2.61
in ECB 4.12
with ENQ 5.26-29,35.14
example of use of 2.52
with GETMAIN 6.14,46.13,47.15
with IDENTIFY 5.4,54.4
requirements 2.51
from STAE 5.176,72.18
return of control
of CPU 2.53-61,2.98-129
(see also RETURN)
of main storage
(see FREEMAIN)
of resource
(see DEQ)
RETURN macro instruction 2.57-60,60.1
examples of 2.59-60
with simple structure load module 2.57-60
returning control
responsibility count 2.127-129
using a branch instruction 2.122-124
using the control program 2.117-129
using RETURN macro instruction 2.57-60
when ATTACH was used 2.118-119
when LINK was used 2.118
without using the control program 2.122-124
returning control in a simple structure 2.53-61
reusability 2.87-97
rname operand of ENQ 5.13-14,35.7
rollout/rollin 6.3
routing codes (with MCS) 5.70,99.1
routing the message to the operator (with MCS) 5.65-75

save area
chaining 2.15,2.18,5.191
description of 2.6-14
flag 2.57,2.59
format 2.7
provision of 2.11-14
register 2.12,2.18
SAVE macro instruction 2.9,61.1
saving registers 2.6-14
providing a save area 2.11-14
save area chaining 2.15,2.18,5.91
save area format 2.7
searching for a usable copy of the load module 2.73-97
effect of DE operand on 2.82-86
effect of EP operand on 2.76-81
effect of EPLOC operand on 2.76-81
order of search 2.73-92
use of BLDL with DE 2.84
SEGID macro instruction 63.1
SEGWT macro instruction 64.1
sequence identifier, calling 5.8
serial execution of a load module 2.28-124
serially reusable load module 2.113-120
restriction on using LINK macro instruction 2.98
using ENQ macro instruction 2.114
serially reusable resource 5.9-12
shared control, (see ENQ macro instruction)
SHSPL operand of ATTACH 3.7,6.28,14.21
(see also main storage management)
SHSPV operand of ATTACH 3.7,6.28,14.20
(see also main storage management)
simple structure 2.25-61
definition of 2.25
passing control with return 2.40-49
passing control without return 2.36-39
returning control 2.53-61
returning control to the control program 2.61
simulator, extended-precision floating-point 5.125-155
SNAP macro instruction 5.188-193
allowing for a variable-length parameter list 9.5
coding in MFT 65.1
coding in MVT 66.1
SPIE macro instruction
coding 69.1
description 5.101-102
with DXR macro instruction 34.1
example 5.106
program interruption control area (PICA) 5.103-106,69.8
program interruption element (PIE) 5.107-108
STAE exit routine 5.166-183
conditions when not executed 5.177
register contents when control received 5.178-179
restriction on use of STAE and ATTACH 5.168
return codes
work area (figure) 5.178
STAE macro instruction
canceling current STAE 5.175,72.9
coding 72.1
example 5.174
exit routine 5.166-183
intercepting abnormal termination 5.166-183
OV operand 5.175,72.10
overriding ABEND 5.159
register contents after execution 5.176

XCTL operand 5.172,5.174,72.13
STAE retry routine 5.182
STAI operand of ATTACH 5.184,14.23
STAI retry routine 5.185
STATUS macro instruction 4.15-16,75.1
STAX macro instruction 5.115
STEP operand
of ABEND 5.161,11.6
of ENQ 5.14
STIMER macro instruction
canceling during ABEND 5.162
coding 77.1
establishing a time interval for a task 5.54-64
example 5.59
specifying how to decrement the interval 5.57
STOP command 5.91-5.94
structure, load module
(see dynamic structure; load module; overlay structure, planned; simple structure)
subpool
creation 6.23-25,6.29
exclusive use 6.25-30
handling
by ATTACH 6.28-31
by GETMAIN 6.28-31
MFT with subtasking 6.18
MFT without subtasking 6.17
under MVT 6.19-31
ownership 6.28
restriction on transfer 6.30
sharing 6.28,6.31
in task communication 6.32
subpool 0 6.18,6.23-28
subpool 240 6.18
subpool 255 6.18
subtasking
MFT systems with 3.25-27
MFT systems without 3.24
subtasks
communication among 4.5-9
creating 3.18
definition of 3.5
hierarchy 4.2-4
priority 3.18-22
termination 4.5-9
SYSABEND DD statement
if omitted 5.190
providing 5.190-192
system log
alternate data set defined 5.86
data sets 5.85-86
definition of 5.85
primary data set defined 5.86
using, via WTL macro instruction 5.87-88
system message blocks (SMBs) 5.77
SYSTEM operand of ENQ 5.14,35.12
SYSUDUMP DD statement
if omitted 5.190
providing 5.190-193
SZERO operand of ATTACH 6.28,14.22

task
communication among 4.5-9
creation 3.1-31
hierarchy 4.1-4
management 4.1-16
priority 3.10-22
signaling task termination 4.5-9
termination 4.5-9
task control block (TCB)
address 3.8
completion code in 4.7
obtaining information from 5.40-46
removal from system 4.5
subtask for 4.5
warning for using with CHAP, EXTRACT, DETACH 4.9
task input/output table (TIOT) address in task control block 5.44
TASK parameter of STIMER 5.57,77.4,78.5
TCB (see task control block)
TIME macro instruction 5.48-53,80.1
BIN operand 5.53,80.5
DEC operand 80.4
MIC operand 80.7
TU operand 5.53,80.6
time slicing 3.23-31
effect on using ATTACH and CHAP 3.29
in MFT with subtasking 3.25-27
in MFT without subtasking 3.24
in MVT 3.28-31
time stamping for the hard copy log 5.83
timing services
date and time of day 5.48-50
interval option 5.47
interval timing 5.54-64
example of interval timing 5.59
time option 5.47
TOD (time-of-day clock) 5.51-52
trace facility, generalized 5.96-99
trace, save area 2.12
trace table 5.194
TTIMER macro instruction
canceling time remaining in a time interval 5.54,82.2
coding 82.1
testing time remaining in a time interval 5.54
TUINTVL (timer unit interval) 5.55,78.10

UNPK instruction
example 5.49
use with time option 5.50
use count, (see responsibility count)

variable type (V) explicit request for main storage 6.13-14,46.6,47.7

wait condition
from ATTACH, LINK, XCTL 2.88
effect of 4.12

from ENQ 5.17-31
from STIMER 5.57
from WAIT 4.10-14
WAIT macro instruction 4.10-14,83.1
WAIT parameter of STIMER 5.57,77.5,78.6
writing to the hard copy log 5.80-84
writing to the operator 5.65-75
using WTO macro instruction 5.65-72
using WTOR macro instruction 5.65,5.73-75
writing to the programmer 5.76-79
writing to the system log 5.85-88
WTL macro instruction 5.87-88,85.1
WTO macro instruction 5.65-72,88.1,89.1
DESC operand 5.72,88.3,89.7
example 5.72
multiple-line form 5.66-69
ROUTCDE operand 5.72,88.3,89.6
used to write to the hard copy log 5.82
used to write to the programmer 5.76-79,88.3
WTOR macro instruction 5.65,5.73-75,92.1,93.1
with abnormal termination 5.162
example 5.74
used to write to the hard copy log 5.82
used to write to the programmer 5.76-79,92.2

XCTL macro instruction 2.121,2.125-129,96.1
and directory entries 6.45
EP, EPLOC, DE operands 2.74
implied request for storage 6.33
in MFT without subtasking 2.91
not using with branch 2.121
passing control without return 2.121-129
protecting against unusable copy 2.122
and responsibility count 2.127-129
similarity to LINK 2.127-128
with main storage hierarchy support 6.54
XCTL operand of STAE 5.172,72.13

2361 Core Storage
hierarchies 6.51
Models 1 and 2 6.51
specifying, in GETMAIN (example) 6.42

GC28-6646-7

OS Supervisor and Macro Instr. Printed in U.S.A. GC28-6646-7

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

IBM System/360 Operating System
Supervisor Services and Macro Instructions

Order No. GC28-6646-7

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

How did you use this publication?

- ☐ As an introduction
- ☐ As a text (student)
- ☐ As a reference manual
- ☐ As a text (instructor)
- ☐ For another purpose (explain)____

____

Please comment on the general usefulness of the book; suggest additions, deletions, and clarifications; list specific errors and omissions (give page numbers):

Cut or Fold Along Line

What is your occupation?____

Number of latest Technical Newsletter (if any) concerning this publication:_______________________

Please include your name and address in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

GC28-6646-7

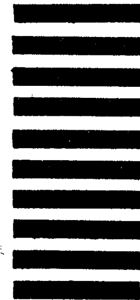
Your comments, please . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Cut or Fold Along Line

Fold   Fold

First Class
Permit 40
Armonk
New York



Business Reply Mail

No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department 636
Neighborhood Road
Kingston, New York 12401

Fold   Fold

OS Supervisor and Macro Instr. Printed in U.S.A. GC28-6646-7

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)