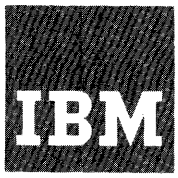
File No. S360-36
Form C28-6541-1

OS

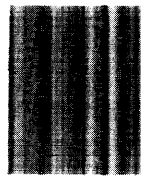


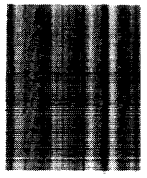
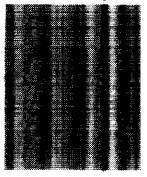
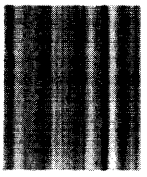
Systems Reference Library

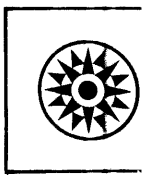
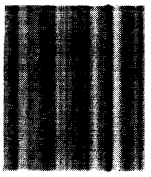
IBM System/360 Operating System

Control Program Services

This publication describes the use of system macro-instructions that request the supervisor, data management, and TESTRAN services of the System/360 control program. It also presents the linkage conventions that have been established for use in the System/360 Operating System.



MAJOR REVISION (April, 1966)
This edition, Form C28-6541-1, obsoletes Form C28-6541-0 and all earlier editions. In addition to incorporating information released in Technical Newsletters N28-2112 and N28-2113, significant changes have been made to the section "Exceptional Condition Handling" and to the Queued and Basic Indexed Sequential Access Methods. This new edition should be reviewed in its entirety.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

A form for readers' comments appears at the back of this publication. It may be mailed directly to IBM. Address any additional comments concerning this publication to the IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602

© by International Business Machines Corporation, 1965, 1966

PREFACE

This publication explains how system macro-instructions are written to request the supervisor, data management, and TESTRAN services of the control program portion of the System/360 Operating System.

The publication is divided into five principal parts: an introduction, descriptions of supervisor macro-instructions and services, descriptions of data management macro-instructions, descriptions of TESTRAN macro-instructions and services, and several appendixes. The appendixes include special reference material on operand forms, operand processing, supplementary macro-instruction forms, dynamic program management, external storage labels, standard status codes, control character codes, and SYSOUT record format.

The introduction describes the various types of system macro-instructions, and explains the notation used in the illustrations of the macro-instruction formats. The introduction also presents the linkage conventions adopted for use in the System/360 Operating System. These are the conventions used in passing control and information from one program or subprogram to another; they are used in all interfaces between the operating system and user programs. It is strongly recommended that they be employed throughout user programs to simplify programmer training and program maintenance.

PREREQUISITE PUBLICATIONS

Knowledge of information contained in the following publications is required for an understanding of this publication:

IBM System/360 Operating System: Introduction, Form C28-6534

IBM System/360 Operating System: Concepts and Facilities, Form C28-6535

IBM System/360 Operating System: Assembler Language, Form C28-6514

IBM System/360 Operating System: Data Management, Form C28-6537

IBM System/360 Operating System: Linkage Editor, Form C28-6538

IBM System/360 Operating System: Job Control Language, Form C28-6539

Knowledge of the full capabilities and techniques of the macro-language, as presented in the Assembler Language publication, is not required for an understanding of this publication. Knowledge of the basic assembler language, however, is required.

CONTENTS

SECTION 1: INTRODUCTION. 11

System Macro-Instructions 11
General Information. 11
Supervisor and Data Management Macro-Instructions 11
TESTRAN Macro-Instructions. 12
Parameters. 12
The Macro Language 13
Macro-Instruction Fields. 13
Types of Macro-Instruction Operands 14
Basic Notation Used to Describe Macro-Instructions 15
Operand Representation. 16
Operands with Value Mnemonics 16
Coded Value Operands. 19
Metasymbols 20
Types of Macro-Instructions. 21
R-Type Macro-Instructions 21
S-Type Macro-Instructions 22
Special Register Notation. 22
Packed Parameters 23

Linkage Conventions 24
Linkage Terminology. 24
Linkage Types. 25
Linkage Registers. 27
Save Area Use. 29
Register Saving and Restoring Responsibilities. 29
Save Area 30
Save Area Chaining. 32
Calling Sequence and Entry Point Identifiers 34
Linkage Interface Responsibilities 34
Called Program Interface in Type I, Type II, and Certain Type IV Linkages. 34
Calling Program Interface in a Type I Linkage Resulting From a Hand-Coded Calling Sequence. 35
Calling Program Interface in Type I, Type II, and Type III Linkages Resulting From Supervisor and Data Management Macro-Instructions 35
Passing Control Information to a Job Step. 36

Macro-Instruction Descriptions. 37

SECTION 2: SUPERVISOR SERVICES 38
Simple Program Management. 41
CALL -- Call a Program (S). 41
SAVE -- Save Register Contents. 44
RETURN -- Return to a Program 46
Overlay Program Management 48
CALL Macro-Instruction in Overlay Management. 49
Branch Instruction in Overlay Management. 49
SEGLD -- Load Segment While Processing (R). 50
SEGWT -- Load Segment Before Further Processing (R) 51
Dynamic Program Management 52
LINK -- Link to a Load Module (S) 52
XCTL -- Transfer Control to a Load Module (S) 56
LOAD -- Load and Retain a Load Module (R) 60
DELETE -- Delete a Retained Load Module (R) 62
IDENTIFY -- Identify an Embedded Entry Point (R). 63
Main Storage Management. 65

GETMAIN -- Allocate Main Storage (R). 65
GETMAIN -- Allocate Main Storage (S). 67
FREEMAIN -- Release Allocated Main Storage (R). 71
FREEMAIN -- Release Allocated Main Storage (S). 72
Task Creation and Management 75
ATTACH -- Create and Attach a Task (S). 75
DETACH -- Remove a Task (R) 84
CHAP -- Change Dispatching Priority (R) 85
EXTRACT -- Extract Selected TCB Fields (S). 87
Task Synchronization 89
WAIT -- Wait for Event (R). 89
WAITR -- Wait for Event and Ready Lower Priority Task (R) . . . 92
POST -- Signal Event Completion (R) 93
ENQ -- Enqueue Request for a Serially Reusable Resource (R) . . 94
DEQ -- Dequeue Request for a Serially Reusable Resource (R) . . 96
Exceptional Condition Handling 97
SPIE -- Specify Program Interruption Exit (S) 97
STAE -- Specify Task Abnormal Exit (R). 101
ABEND -- Terminate a Task Abnormally (R). 102
CHKPT -- Checkpoint a Job Step (R). 105
General Services 107
TIME -- Request Time and Date (R) 107
STIMER -- Set Interval Timer (R). 108
TTIMER -- Test Interval Timer (R) 111
WTO -- Write to Operator (S). 111
WTOR -- Write to Operator with Reply (S). 112
WTL -- Write to Log (S) 113

SECTION 3: DATA MANAGEMENT SERVICES 114
Direct Access Device Considerations 114
Volume Switching. 116
General Service Macro-Instructions 117
DCB -- Define a Control Block for Input/Output Operations . . .117
DCBD -- Provide Symbolic Names for a Data Control Block (DCB) .119
OPEN -- Prepare the Data Control Block for Processing (S) . . .122
CLOSE -- Disconnect Data Set from User's Problem Program (S). .124
FEOV -- Force End of Volume (R) 127
GETPOOL -- Get a Buffer Pool (R). 127
FREEPOOL -- Free a Buffer Pool (R). 128
BUILD -- Build a Buffer Pool (R). 129
GETBUF -- Get a Buffer From a Pool (R). 130
FREEBUF -- Return a Buffer to a Pool (R). 131
Queued Sequential Access Method (QSAM) 132
DCB - Define Data Control Block for QSAM. 134
GET -- Locate Mode (R). 141
GET -- Move Mode (R). 143
GET -- Substitute Mode (R). 144
PUT -- Locate Mode (R). 145
PUT -- Move Mode (R). 146
PUT -- Substitute Mode (R). 147
PUTX -- Update Mode (R) 149
PUTX -- Output Mode (R) 150
RELSE -- Release an Input Buffer (R). 152
TRUNC -- Truncate an Output Buffer (R). 152
CNTRL -- Control a Printer or Stacker (R) 153
PRTOV -- Test for Printer Carriage Overflow (R) 154
Basic Sequential Access Method (BSAM). 155
DCB -- Define Data Control Block for BSAM 156
READ -- Read a Block (S). 164
WRITE -- Write a Block (S). 166
WRITE -- Update a Block (S) 167
CHECK -- Wait for and Test Completion of Read or Write Operation (R). 169
CLOSE (TYPE=T) -- Temporarily Disconnect a Data Set from Problem Program (S). 171

NOTE -- Provide Position Feedback (R) 172
POINT -- Position to a Block (R). 174
BSP -- Backspace a Block (R). 175
PRTOV -- Test for Printer Carriage Overflow (R) 176
CNTRL -- Control On-Line Input/Output Devices (R) 177
WRITE -- Create a Direct Organization Data Set - Format-F Records (S). 179
WRITE -- Create a Direct Organization Data Set - Format-U or -V Records or a Capacity Record (S). 180
Basic Partitioned Access Method (BPAM) 182
Partitioned Data Organization 183
Partitioned Organization Directory Format 183
DCB -- Define Data Control Block for BPAM 186
FIND -- Position to Member of Partitioned Data Set (R). 189
BLDL -- Build List (R). 190
STOW -- Manipulate Partitioned Data Set Directory (R) 192
Queued Indexed Sequential Access Method (QISAM). 194
DCB - Define Data Control Block for QISAM - Load Mode 194
QISAM Load Mode Buffer Requirements 199
PUT -- Move Mode (R). 200
PUT -- Locate Mode (R). 203
DCB -- Define Data Control Block for QISAM - Scan Mode. 204
QISAM Scan Mode Buffer Requirements 206
SETL -- Specify Start of Sequential Retrieval (R) 207
ESETL -- End Sequential Retrieval (R) 209
GET -- Locate Mode (R). 209
GET -- Move Mode (R). 210
QISAM Scan Mode Work Area Requirements. 211
PUTX -- Update Mode (R) 211
RELSE - Release Current Input Buffer (R). 213
Basic Indexed Sequential Access Method (BISAM) 213
DCB -- Define Data Control Block for BISAM. 214
READ -- Retrieve a Logical Record (S) 218
BISAM Area Requirements 221
WRITE -- Write a Logical Record (S) 222
FREEDBUF -- Free Dynamically Obtained Buffer (R). 223
Basic Direct Access Method (BDAM). 224
DCB -- Define Data Control Block for BDAM 225
READ -- Read a Block (S). 229
WRITE -- Write a Block (S). 233
RELEX -- Release Exclusive Control (R). 237
FREEDBUF -- Free Dynamically Obtained Buffer (R). 238
Queued Telecommunication Access Method (QTAM). 239
Message Processing Routines 239
DCB -- Define QTAM Data Control Block 239
GET -- Obtain Next Record (R) 241
PUT -- Put Next Record (R). 242

SECTION 4: TESTRAN SERVICES. 243

TESTRAN Operation 244

TESTRAN Macro-Instruction Statement Format. 245

TESTRAN Macro-Instructions. 245
Macro-Instruction Descriptions 247
DUMP DATA -- Record Main Storage. 247
DUMP CHANGES -- Record Main Storage and Identify Changes. . . .249
DUMP MAP -- Record Storage Map. 251
DUMP TABLE -- Record System Table 251
DUMP PANEL -- Record Registers and PSW. 252
DUMP COMMENT -- Record Comment. 253
TRACE FLOW -- Record Program Transfers. 253
TRACE CALL -- Record Execution of CALL Macro-Instructions . . .255
TRACE REFER -- Record Storage References. 256

TRACE STOP -- Suspend Traces. 257
TEST OPEN -- Initiate Testing 258
TEST AT -- Perform Testing at Problem Program Address 260
TEST DEFINE -- Define Flags or Counters 261
TEST WHEN -- Alter Test Sequence When Condition or Relationship Occurs. 262
TEST ON -- Alter Test Sequence on Counter Interval. 263
TEST CLOSE -- Terminate Testing 265
GO TO -- Encounter TESTRAN Macro-Instruction. 265
GO IN -- Enter TESTRAN Subroutine 266
GO OUT -- Return from TESTRAN Subroutine. 266
GO BACK -- Return to Problem Program. 267
SET FLAG -- Assign Condition to Flag. 268
SET COUNTER -- Assign Value to Counter. 269
SET VARIABLE -- Assign Value to Storage or Register 269
Notes on Usage 270
Keyword Modifiers 270
Address Specification 273
TEST OPEN Macro-Instructions. 273
TEST CLOSE Macro-Instructions 274
Editing Restrictions. 274
Improperly Coded Macro-Instructions 275

Edited Output Formats 275
Standard Page Heading 277
Output Lines for DUMP DATA and DUMP CHANGES 277
Output Lines for DUMP MAP 279
Output Lines for DUMP TABLE 279
Output Lines for DUMP PANEL 280
Output Lines for DUMP COMMENT 281
Initial Trace Output Lines. 282
Output Lines for TRACE FLOW 283
Output Lines for TRACE CALL 286
Output Lines for TRACE REFER. 287
Output Line for TRACE STOP. 290
Output Lines for TEST OPEN. 291
Output Line for TEST AT 292
Output Line for TEST CLOSE. 292
Output Lines for Other Encountered Control Macro-Instructions .293
Error Message Lines 294

Sample Test Program and Test Output 294

Job Organization. 300

APPENDIX A: OPERAND FORMS. 303

Descriptions of Operand Forms 303
Relocatable expression 304
Implied Address. 304
Explicit Address 305
Absolute Expression. 306
Register Notation. 306
TESTRAN Register Notation. 307
Character Constant 308
Data Attribute Notation. 308

Operand Processing. 309

APPENDIX B: L AND E FORMS OF S-TYPE MACRO-INSTRUCTIONS 312

L- and E-Form Macro-Expansions. 312

Use of L- and E-Form Macro-Instructions 312
The MF Keyword Operand 313

Operand Forms Used in L- and E-Form Macro-Instructions 313
Operand Combinations 314
Ordinary and Special Operand Requirements. 315

APPENDIX C: DYNAMIC PROGRAM MANAGEMENT 317

Contents Control. 317
Reusability. 317
Libraries. 317
Pack Areas 318
Contents Directory 318
Load Module Acquisition Procedures 319

Use of the LOAD Macro-Instruction 320
Reenterable Module From the Link Library 320
Reenterable Module From a Job Library or Private Library 320
Serially Reusable Module From Any Library. 321
NonReusable Module From Any Library. 321

Use Of The Identify Macro-Instruction 321

Use of the LINK, XCTL, and ATTACH Macro-Instructions. 322

APPENDIX D: EXIT LIST DESCRIPTION. 323

APPENDIX E: SECONDARY STORAGE STANDARD LABEL FORMATS. 327

Standard Magnetic Tape Labels 327
Volume Label Group 327
Initial Tape Volume Label Format. 327
Additional Volume Labels Format 328
Data Set Header Label Group. 328
Data Set Header 1 Label Format. 328
Data Set Header 2 Label Format. 330
User Header Label Group. 331
User Header Label Format. 331
Data Set Trailer Label Group 331
User Trailer Label Group 332

Direct-Access Volume Labels 332
Volume Label Group 332
Data Set Control Block Group 332
User Header and Trailer Label Groups 332

APPENDIX F: CONTROL CHARACTERS AND SYSOUT WRITER 333

Control Characters. 333

Machine Code. 333

Extended ASA Code 333

SYSOUT Writers. 334

APPENDIX G: STANDARD STATUS INFORMATION. 335

INDEX 337

FIGURES

Figure 1. Linkages in LINK Macro-Instruction Execution 26
Figure 2. Branching Instructions 50
Figure 3. Format of the Event Control Block (ECB). 91
Figure 4. Format of the Queue Control Block (QCB). 95
Figure 5. Format of the Program Interruption Control Area (PICA) . . 99
Figure 6. Format of the Program Interruption Element (PIE) 99
Figure 7. Sample Test Program. 295
Figure 8. Sample Test Output 298

TABLES

Table 1. Use of Value Mnemonics by Groups of System Macro-Instructions 18
Table 2. Linkage Registers 27
Table 3. Save Area Words and Contents in Calling Programs. 31
Table 4. Services Affected by Including or Excluding Control Program Options. 40
Table 5. Supervisor Actions Upon Subtask Termination 80
Table 6. Data Management Exits 118
Table 7. Magnetic Tape Positions - QSAM and BSAM 125
Table 8. Factors Determining Magnetic Tape Positioning - QSAM and BSAM 126
Table 9. Buffering and Modes of GET-PUT. 133
Table 10. DEN Values 136
Table 11. QSAM Buffer Acquisition and Data Control Block Field Requirements 139
Table 12. Error Options for QSAM 141
Table 13. Register Contents Upon Entry to SYNAD Routine. 142
Table 14. Acceptable Record Formats and Corresponding Buffering Techniques for QSAM and the PUTX Macro-Instruction 151
Table 15. DEN Values 158
Table 16. BSAM and BPAM Buffer Acquisition and Data Control Block Field Requirements 162
Table 17. Format of the Data Event Control Block 164
Table 18. Register Contents Upon Entry to SYNAD Routine. 170
Table 19. Magnetic Tape Temporary Positions - BSAM 172
Table 20. QISAM Buffer Acquisition and Data Control Block Field Requirements 198
Table 21. Contents of Exceptional Condition (DCBEXCD) Fields of Data Control Block -- QISAM Load and Scan Modes. 201
Table 22. Register Contents Upon Entry to SYNAD - QISAM Load and Scan Mode. 203
Table 23. Type Operand for SETL Macro-Instruction. 208
Table 24. BISAM Buffer Acquisition and Data Control Block Field Requirements 217
Table 25. Format of Data Event Control Block for BISAM 218
Table 26. Contents of Exceptional Condition Code Byte, Data Event Control Block - BISAM. 219
Table 27. BDAM Buffer Acquisition and Data Control Block Field Requirements 228
Table 28. Data Event Control Block for BDAM. 229
Table 29. READ Macro-Instruction Type Operand Values for BDAM. . . .230
Table 30. Exception Condition Bits for BDAM. 232
Table 31. WRITE Macro-Instruction Type Operand Values for BDAM . . .235
Table 32. Message Type Byte Definition Chart 242
Table 33. Forms of the TESTRAN Macro-Instructions. 246
Table 34. Common Keyword Operands and Their Usage. 247
Table 35. Printing Formats for Data Types. 276
Table 36. Job Control Statements Required for Assembly, Linkage Editing, Program Testing, and Output Editing 300
Table 37. Operand Forms and Related Value Mnemonics. 303
Table 38. Data Attribute Specifications. 309
Table 39. Program Management in Type II Linkages 322
Table 40. Format and Contents of an Exit List. 323
Table 41. Control Program Response to an Edit Routine Return Code. 324
Table 42. Exit List. 325
Table 43. Label Exits. 326
Table 44. DEN Values 330

The control program provides a comprehensive set of services. These services can be requested directly in a program written in the assembler language, or indirectly in a program written in a higher level language. This set of services is subdivided in this publication, as follows:

- Supervisor services, which provide linkage between programs, obtain and release allocated main storage, manage tasks, set and test an interval timer, etc.
- Data management services, which allow conventional and advanced forms of processing of data sets existing on various types of external storage devices. These services consist of several access methods applicable to various data set organizations.
- TESTRAN services, which provide many convenient ways by which the programmer can test and debug his program.

Some of the above services are provided by routines that are an integral part of the resident control program. The remainder are provided by routines that are loaded into main storage by the control program only when required.

SYSTEM MACRO-INSTRUCTIONS

Control program services are requested by means of system macro-instructions included in the user's problem program.

GENERAL INFORMATION

System macro-instructions are processed by the assembler program using macro-definitions supplied by IBM and placed in the macro-library at system generation time.

The processing of a macro-instruction by the assembler is called the expansion of the macro-instruction. This processing results in fields of data and executable instructions, called the macro-expansion. Elements of a macro-expansion are referred to in terms of their assembler language statement equivalents.

Supervisor and Data Management Macro-Instructions

The macro-expansion of a supervisor or data management macro-instruction is in-line in the user's problem program. The macro-expansion contains either a supervisor call (SVC) instruction or a branch instruction, which gives control to the control program routine that is to perform the requested service. At execution time, the macro-expansion passes fields of information to the control program routine to specify the exact nature of the service to be performed. These data fields are called parameters; they are passed in either registers or a data area, as follows:

- In certain macro-instructions, parameters are passed in registers called parameter registers. The macro-expansion can contain load address (LA) instructions that form parameters in parameter registers at execution time, and it can contain instructions that load parameter registers from registers loaded by the user's problem program. The user's problem program can also load parameter registers directly. Registers 0 and 1 are used as parameter registers.

- In macro-instructions that do not pass parameters in registers, parameters are passed in a data area called a parameter list. Parameters can be assembled in the list as constants, and they can be stored in the list by the macro-expansion from registers loaded by the user's problem program. The macro-expansion loads register 1 or 15 with the address of the list, and the control program routine uses this register to refer to the list. In this use, register 1 or 15 is called the parameter list register.

TESTRAN Macro-Instructions

The macro-expansion of a TESTRAN macro-instruction is out-of-line in a special control section that consists of a single SVC instruction followed by a series of constants. When this control section is given control at execution time, TESTRAN service routines are fetched into main storage and the macro-expansions contained in the control section are interpreted. The service routines, collectively called the TESTRAN interpreter, insert SVC instructions in the user's problem program as designated by the macro-expansions. The TESTRAN interpreter saves the displaced user's instructions for execution in their proper sequence. The user's problem program is then resumed. When inserted SVC instructions are subsequently executed, the macro-expansions are again interpreted and the requested services are performed.

Data produced by TESTRAN macro-instructions is passed to the TESTRAN editor, a processing program that edits and prints data in the format defined by the source program.

Parameters

Each parameter resulting from the expansion of a supervisor or data management macro-instruction is either an address or a value; this is true whether the parameter is in a register or a list.

ADDRESS PARAMETER: An address parameter is the standard 24-bit address. It is always located in the three low-order bytes of either a parameter register or a full-word in a parameter list. The full-word in the parameter list is aligned on a full-word boundary.

The high-order byte in either the parameter register or the full-word in the parameter list contains all zeros. Any exceptions to this rule are stated in individual macro-instruction descriptions.

An address parameter is always an effective address. The control program is never given a 16- or 20-bit explicit address (of the form D(B) or D(X,B)) and then required to form an effective address. If an effective address must be formed dynamically, it is formed either by the macro-expansion or before the macro-instruction is issued.

VALUE PARAMETER: A value parameter is a field of data other than an address. It is of variable length, and is usually in the low-order bits of either a parameter register or a full-word in a parameter list. The full-word in the parameter list is aligned on a full-word boundary. Unless explicitly stated otherwise, a parameter has binary format.

The high-order unused bits in either the parameter register or the full-word in the parameter list contain all zeros. Any exceptions to this rule are stated in individual macro-instruction descriptions.

Certain value parameters are placed in a register or a full-word along with another parameter, which can be either an address or a value parameter. In this case, a value parameter will be in other than the low-order bits. Two parameters in the same register or full-word are called packed parameters.

Certain value parameters are longer than a full-word. For example, a parameter might consist of the characters of an eight-character symbol, or it might consist of eight unpacked decimal digits. This kind of parameter is passed to the control program only in a parameter list.

OPERANDS: Parameters are specified by operands in the macro-instruction. An address parameter can result from a relocatable expression or, in certain macro-instructions, from an implied or explicit address. A value parameter can result from an absolute expression or a specific character string. Address and value parameters can both be specified by operands written as an absolute expression enclosed in parentheses; this operand form is called register notation. The value of the expression designates a register into which the specified parameter must be loaded by the user's problem program. The contents of this register are then placed in either a parameter register or a parameter list by the macro-expansion.

THE MACRO LANGUAGE

Certain of the rules for writing system macro-instructions, and the terminology used, are discussed in the following paragraphs. This information is partly a subset of that in the publication IBM Operating System/360: Assembler Language, but contains certain rules that apply to only the system macro-instructions.

Macro-Instruction Fields

System macro-instructions, like assembler instructions, are written in the following general format:

Name	Operation	Operand
A symbol or blank	Mnemonic operation code	Zero or more operands separated by commas

The name field of the macro-instruction can contain a symbol. A symbol written in this field can be used to refer to the first assembler language statement (other than a CNOP) resulting from the macro-instruction.

The operation field contains the mnemonic operation code of the macro-instruction.

The operand field can contain a list of operands separated from one another by commas. The operands, in conjunction with the mnemonic operation code, specify the particular service requested by the macro-instruction.

If a macro-instruction format permits a blank operand field, any comment must be preceded by a comma followed by a blank in order to delimit the operand field.

Types of Macro-Instruction Operands

The programmer writes operands in a system macro-instruction to specify the exact nature of the service to be performed. When the macro-instruction is processed by the assembler program, operands result in such elements of the macro-expansion as:

- Either a constant parameter or executable instructions that form a parameter at execution time. This occurs in supervisor or data management macro-instructions.
- A constant in a special control section. This occurs in TESTRAN macro-instructions.

Operands are of two types: positional and keyword.

POSITIONAL OPERANDS: A positional operand is written as a string of characters. This character string can be an expression, an implied or explicit address, or some special operand form allowed in a particular macro-instruction. (Refer to "Operand Representation.")

Positional operands must be written in a specific order. If a positional operand is omitted and another positional operand is written to the right of it, the comma that would normally have preceded the omitted operand must be written. This comma should be written only if followed by a positional operand; it should not be written if it would be followed by a keyword operand or a blank.

In the following examples, EX1 has three positional operands. In EX2, the second of three positional operands is omitted, but must still be delimited by commas. In EX3, the first and third operands are omitted; no comma need be written to the right of the second operand.

```
EX1     EXAMP   A,B,C
EX2     EXAMP   A,,C
EX3     EXAMP   ,B
```

KEYWORD OPERANDS: A keyword operand is written as a keyword immediately followed by an equal sign and an optional value.

A keyword consists of one through seven letters and digits, the first of which must be a letter. It must be written exactly as shown in a macro-instruction description.

An optional value is written as a character string in the same way as a positional operand.

Keyword operands can be written in any order, but they must be written to the right of any positional operands in the macro-instruction.

In the following examples, EX1 shows two keyword operands. EX2 shows the keyword operands written in a different order and to the right of positional operands. In EX3, the second and third positional operands are omitted; they need not be delimited by commas, because they are not followed by any positional operands.

```
EX1       EXAMP  KW1=X,KW2=Y
EX2       EXAMP  A,B,C,KW2=Y,KW1=X
EX3       EXAMP  A,KW1=X,KW2=Y
```

OPERAND SUBLISTS: A positional operand or the optional value of a keyword operand can be written as a sublist, if this is specified by a particular macro-instruction description.

A sublist consists of one or more operands, of the form of a positional operand, separated by commas and enclosed in parentheses. The entire sublist, including the parentheses, is considered to be one positional operand or the optional value of a keyword operand. For example:

```
(A,B,C)
(A)
 KW1=(A,B,C)
```

Note that, in the second example above, the sublist consists of only one operand.

When a supervisor or data management macro-instruction description shows that an operand or optional value is to be written as a sublist, the enclosing parentheses must be written, even if there is only one element in the sublist. The parentheses that designate a sublist are in addition to parentheses used in register notation.

When a TESTRAN macro-instruction description shows that an operand or optional value is to be written as a sublist, the programmer can either write the enclosing parentheses or omit them when there is only one element in the sublist. This is because the form of register notation used in TESTRAN macro-instructions is different from the form used in supervisor and data management macro-instructions.

REQUIRED AND OPTIONAL OPERANDS: Certain operands are required in a macro-instruction, if the macro-instruction is to make a meaningful request for a control program service. Other operands are optional, and can be omitted. Whether an operand is required or optional is indicated in the macro-instruction descriptions.

BASIC NOTATION USED TO DESCRIBE MACRO-INSTRUCTIONS

System macro-instructions are presented in this publication by means of macro-instruction descriptions, each of which contains an illustration of the macro-instruction format. This illustration is called a format description. An example of a format description is as follows:

Name	Operation	Operand
[symbol]	EXAMP	$name_1$-value mnemonic,$name_2$-CODED VALUE ,CODED VALUE ,KEYWD1=value mnemonic,KEYWD2=CODED VALUE

Operand representations in format descriptions contain the following elements:

- An operand name, which is a single mnemonic word used to refer to the operand. In the case of a keyword operand, the keyword is the name. In the case of a positional operand, the name is merely a referent, or it is a coded value (see below). In the above format description, $name_1$, $name_2$, CODED VALUE (in the third operand), KEYWORD1, and KEYWORD2 are operand names.

- A value mnemonic, which is a mnemonic used to indicate how the operand should be written, if it is not written as a coded value. For example, addr is a value mnemonic that specifies that an operand or optional value is to be written as either a relocatable expression or register notation.

- A coded value, which is a character string that is to be written exactly as it is shown. For example, TASK is a coded value.

The format description also specifies when single operands and combinations of operands should be written. This information is indicated by notational elements called metasymbols. For example, in the preceding format description, the brackets around symbol in the name field indicate that a symbol in this field is optional.

Operand Representation

Positional operands are represented in format descriptions in one of three ways:

- By a three-part structure consisting of an operand name, a hyphen, and a value mnemonic. For example: $name_1$-addr.

- By a three-part structure consisting of an operand name, a hyphen, and a coded value. For example: $name_1$-TASK.

- By a coded value. For example: TASK.

Keyword operands are represented in format descriptions in one of two ways:

- By a three-part structure consisting of a keyword, an equal sign, and a value mnemonic. For example: KEYWD1=addr.

- By a three-part structure consisting of a keyword, an equal sign, and a coded value. For example: KEYWD1=TASK.

The most significant characteristic of an operand representation is whether a value mnemonic or a coded value is used; these two cases are discussed below.

Operands with Value Mnemonics

When a keyword operand is represented by:

```
KEYWORD=value mnemonic
```

the programmer first writes the keyword and the equal sign, and then a value of one of the forms specified by the value mnemonic.

When a positional operand is represented by:

name-value mnemonic

the programmer writes only a value of one of the forms specified by the value mnemonic. The operand name is merely a means of referring to the operand in the format description; the hyphen simply separates the name from the value mnemonic. Neither is written.

The following general rule applies to the interpretation of operand representations in a format description: when the operand is written, anything shown in upper-case letters must be written exactly as shown; anything shown in lower-case letters is to be replaced with a value provided by the programmer. Thus, in the case of a keyword operand, the keyword and equal sign are written as shown, and the value mnemonic is replaced. In the case of a positional operand, the entire operand representation is replaced.

VALUE MNEMONICS: The value mnemonics listed below specify most of the allowable operand forms that can be written in system macro-instructions. Other value mnemonics, which are rarely used, are defined in individual macro-instruction descriptions.

- symbol - the operand can be written as a symbol.
- relexp - the operand can be written as a relocatable expression.
- addr - the operand can be written as (1) a relocatable expression, or (2) register notation designating a register that contains an address in its three low-order bytes and all-zeros in its high-order byte. Register notation is written as an absolute expression that begins with a left parenthesis and ends with a right parenthesis. (These parentheses are not necessarily paired.) The value of the absolute expression is the number of the designated register. The designated register must be one of the registers 2 through 12, unless special register notation is used. (Refer to "Special Register Notation.")
- addrx - the operand can be written as (1) an indexed or nonindexed implied or explicit address, or (2) register notation designating a register that contains an address in its three low-order bytes and all-zeros in its high-order byte. An explicit address must be written as in the RX form of an assembler language instruction.
- addx - the operand can be written as an indexed or nonindexed implied or explicit address. An implied address cannot be written as a literal. An explicit address must be written as in the RX form of an assembler language instruction.
- adval - the operand can be written as (1) an indexed or nonindexed implied or explicit address, or (2) TESTRAN register notation for a register that contains a value. TESTRAN register notation is written as the letter G (for general register) or the letter F (for floating-point register) followed by an absolute symbolic term, or an integer, enclosed in single quotation marks. The value of the symbolic term or integer is the number of the designated general or floating-point register. There is no restriction on which register is designated. An explicit address must be written as in the RX form of an assembler language instruction. (If an implied address is written as a literal, the address will refer to a constant contained in the macro-expansion.)
- integer - the operand can be written as an integer (a decimal self-defining term).

- absexp - the operand can be written as an absolute expression.

- value - the operand can be written as (1) an absolute expression, or (2) register notation designating a register that contains a value in its low-order bits and all-zeros in its unused high-order bits.

- text - the operand can be written as a character constant as in a DC data definition instruction. (The format description shows explicitly that the character constant is to be enclosed in single quotation marks.)

- code - the operand can be written as one of a large set of coded values; these values are defined in the macro-instruction description.

- tls - the operand can be written as data attribute notation. Data attribute notation is written as the type and modifier subfields of a DC or DS data definition statement, and specifies type, length, and/or scale attributes for data processed by the TESTRAN interpreter and editor.

The subset of value mnemonics used by each group of system macro-instructions, and the use of the corresponding operands, is shown in Table 1.

Table 1. Use of Value Mnemonics by Groups of System Macro-Instructions

Operand Use	Group of System Macro-Instructions		
	Supervisor	Data Management	TESTRAN
Specifies an address	symbol relexp addr addrx	symbol relexp addr addrx	symbol relexp addx adval
Specifies a value	integer absexp value	absexp value	integer adval
Specifies other information	text code	code	symbol text tls

Additional information on operand forms and operand processing is given in Appendix A.

The following example illustrates the use of value mnemonics in a format description:

Name	Operation	Operand
[symbol]	EXAMP	$name_1$-symbol,$name_2$-addrx,KEYWD1=absexp ,KEYWD2=value

Each of the four operands shown can be written in any one of the forms specified by its value mnemonic.

In the following examples, the macro-instructions are written as directed by this format description.

In EX1, the $name_1$ operand is a symbol, the $name_2$ operand is an implied address, and the KEYWD1 and KEYWD2 optional values are absolute expressions.

In EX2, the $name_2$ operand is an indexed implied address.

In EX3, the $name_2$ operand is an explicit address, and the KEYWD2 optional value is register notation. When the macro-instruction is issued, register 10 should contain the parameter specified by the KEYWD2 operand.

```
EX1     EXAMP   ALPHA,PAYROLL+8,KEYWD1=25,KEYWD2=100
EX2     EXAMP   ALPHA,PAYROLL+8(5),KEYWD1=W50,KEYWD2=4*W50
EX3     EXAMP   ALPHA,40(0,5),KEYWD1=W50,KEYWD2=(10)
```

Coded Value Operands

Some operands are not represented in format descriptions by value mnemonics. Instead, they are represented by one or more upper-case character strings that show exactly how the operand should be written. These character strings are called coded values, and the operands for which they are written are called coded value operands.

A coded value operand results in either a specific value parameter or a specific sequence of executable instructions.

When a positional operand can be written as only one coded value, the operand is shown simply as the coded value; an additional lower-case operand name is not used. For example, a positional operand could be represented by:

```
TASK
```

A keyword operand could be represented by:

```
KEYWORD=TASK
```

If a positional operand can be written as any one of two or more coded values, an additional lower-case operand name may or may not be used. The choice of which is done is determined by whether or not a name can be meaningfully used to refer to all values of the operand. For example, a positional operand could be shown as either of the following:

```
{TASK|REAL}
mode-{TASK|REAL}
```

In both of the above examples, the braces indicate that the coded values are grouped together in one operand representation, and the vertical stroke indicates that either one of the coded values can be written. The braces and vertical strokes are metasymbols.

Metasymbols

Metasymbols are symbols that convey information to the programmer, but are not written by him. They assist in showing the programmer how and when an operand should be written. The metasymbols used in this publication are:

1. | This is a vertical stroke and means "or." For example, A|B means either the character A or the character B. Alternatives are also indicated by being aligned vertically (as shown in the next paragraph).

2. { } These are braces and denote grouping. They are used most often to indicate alternative operands. For example:

   {TASK|REAL}

   $\left\{\begin{matrix}\text{TASK}\\\text{REAL}\end{matrix}\right\}$

   The two examples above are equivalent; either TASK or REAL must be written.

3. [ ] These are brackets and denote options. Anything enclosed in brackets can be either omitted or written once in the macro-instruction. For example:

   [TASK]

   [<u>TASK</u>|REAL]

   $\left[\begin{matrix}\underline{\text{TASK}}\\\text{REAL}\end{matrix}\right]$

   The second and third examples above are equivalent; TASK, or REAL, or neither can be written. The underlining indicates that, if neither is written, TASK is assumed. Braces used for grouping inside brackets are redundant.

4. ... This is an ellipsis. It denotes occurrence of the preceding syntactical unit one or more times in succession. A syntactical unit is any combination of operand representations, commas, parentheses, and metasymbols, enclosed in braces. For example:

   {symbol,}...

   The above example indicates that a symbol followed by a comma can be written any number of times, but it must be written at least once. The braces denote grouping, and are the extremities of the syntactical unit to which the ellipsis refers.

The following example shows metasymbol use in a format description:

Name	Operation	Operand
[symbol]	EXAMP	({abc-addr,[def-{TASK|REAL}],}...)

The enclosing parentheses specify a sublist.

The outer pair of braces followed by the ellipsis indicates that the sublist can consist of one or more occurrences of the syntactical unit bounded by the braces.

The comma to the left of the rightmost brace is required to make the format description correct (since all operands except the first must be preceded by a comma). A trailing comma is unnecessary and must not be written.

The brackets indicate that the def operand within them is optional. If the def operand is used, it is written as either TASK or REAL.

The comma to the left of the leftmost bracket is not enclosed by the bracket, because it must be written if any positional operand is written to the right of it. For example, the operand field might contain:

```
(DCB1,,DCB2)
```

indicating that two abc operands, DCB1 and DCB2, are written with no def operands.

TYPES OF MACRO-INSTRUCTIONS

Most supervisor and data management macro-instructions are referred to as being either R type (register) or S type (storage). An R-type macro-instruction passes parameters to the control program by means of parameter registers; an S-type macro-instruction passes them by means of a parameter list.

A few supervisor and data management macro-instructions do not pass control to the control program. For example, the SAVE macro-instruction results in instructions in the user's problem program that completely perform the requested service. Similarly, the DCB macro-instruction results in only a data area containing constant parameters. These macro-instructions are neither R type nor S type; they are referred to simply as macro-instructions.

R-Type Macro-Instructions

An R-type macro-instruction is used when only one, two, or three[1] parameters are to be passed to the control program. The parameters are passed in register 0 or 1, or both. This results in good performance because:

- A typical R-type macro-expansion consists of fewer executable instructions than would be required if the one to three parameters were passed in a list.
- The user's problem program can often be written so that parameters already exist in registers when the macro-instruction is issued. In this case, instructions that refer to storage are not required in the macro-expansion. If a parameter exists in a register other than a parameter register, a load register (LR) instruction that loads the correct parameter register is part of the macro-expansion.

---

[1]When an R-type macro-instruction has three parameters, two or three of them can be packed into one parameter register. (Refer to "Packed Parameters.")

(Note that if the register contains the address of data, a load instruction is part of the macro-expansion.)

Address operands can be written in R-type macro-instructions as implied or explicit addresses, or by using register notation.

S-Type Macro-Instructions

An S-type macro-instruction is used when three or more parameters are to be passed to the control program. In this case, the parameters are passed in a parameter list. This allows the macro-instruction to be used with no noticeable effect on the contents of most of the registers used by the user's program. (The contents of registers 2 through 13 are not disturbed. Register conventions are described in "Linkage Conventions.") The system macro-instructions pass three or more parameters by list rather than by register because, if large numbers of parameters were passed by registers, the user's problem program might need instructions to save register contents before execution of such a macro-instruction, and then to restore them afterwards.

Use of S-type macro-instructions simplifies the writing of programs. The programmer need not know the identities of registers used by macro-instructions, and he need not plan ahead to achieve the optimum position of parameters in registers.

Address operands in the standard form of S-type macro-instructions can be written only as relocatable expressions or by using register notation. However, implied and explicit addresses can be written if nonstandard forms of the macro-instructions are used; these are called the L and E forms.

The L and E forms of S-type macro-instructions allow a single parameter list to be used by two or more macro-instructions that request the same general control program service. The parameters in the list can be modified to request a specific service each time a macro-instruction is executed. The L and E forms also allow an S-type macro-instruction to be used in a reenterable program even when its parameter list is modified at execution time. Refer to Appendix B for more information about these macro-instruction forms.

SPECIAL REGISTER NOTATION

If an operand of an R-type macro-instruction is written using register notation, the resulting macro-expansion loads the parameter contained in the designated register into either parameter register 0 or parameter register 1.

For example, if an operand is written as (R45), and if the corresponding parameter is to be passed to the control program in register 0, the macro-expansion could contain the instruction LR 0,(R45), or L 0,0(0,R45), or L 0,0(R45,0). (The parentheses are not always removed in the macro-expansion. They have no effect on the action of the assembler program.)

The user's problem program can load parameter registers directly, before execution of the macro-expansion; this is called preloading. The programmer specifies that preloading will occur by writing an operand as either (0) or (1); this is called special register notation.

This notation is special for two reasons:

- The register notation designation of registers 0 and 1 is generally not allowed.

- The designation must be made by the specific three characters (0) or (1), rather than by the general form of an absolute expression enclosed in parentheses. For example, even though the absolute expression R45 could be equated to 0, (R45) must not be written instead of (0) when special register notation is intended. If this were done, the macro-expansion would at least contain a useless LR 0,(R45) instruction; in certain cases, the macro-expansion would contain an undesired L 0,0(0,R45) (or L 0,0(R45,0)) instruction, and would result in an improperly loaded parameter register.

The format description of an R-type macro-instruction shows whether special register notation can be used, and for which operands. This is demonstrated by the following format description:

Name	Operation	Operand
[symbol]	EXAMP	{abc-addrx / (1)}, {def-addrx / (0)}

Both operands can be written in the addrx forms, and therefore can be written using register notation. Ordinary register notation indicates that the parameter register should be loaded from the designated register by the macro-expansion. The format description also shows that the abc operand can be written as (1), and the def operand can be written as (0). If either of these special register notations is used, the user's problem program must have loaded the designated parameter register before execution of the macro-instruction.

Special register notation can also be used to write the optional value of a keyword operand. The operand appears in a format description as shown in the following example:

ABC={addrx / (1)}

Packed Parameters

Certain R-type macro-instructions use registers 0 and 1 to pass three parameters to the control program. Two or three parameters are loaded into one parameter register; these are called packed parameters. The user's problem program can preload packed parameters, provided that it preloads both of them. The fact that preloading is possible and the required special register notation are shown in the following example:

Name	Operation	Operand
[symbol]	EXAMP	{abc-addrx / (1)}, {def-value,ghi-value / (0)}

For this example, the operand descriptions could state that the def parameter can be preloaded into the two high-order bytes of register 0, and the ghi parameter can be preloaded into the two low-order bytes of register 0. Then, if (0) is written for the second operand of the macro-instruction, it specifies both the def and ghi parameters, and both these parameters must be preloaded into register 0. Note that ordinary register notation (designating one of the registers 2 through 12) can be written for the def operand or the ghi operand, or both, if these operands are written separately. Where ordinary register notation is used, the specified value must be in the rightmost two bytes of the designated register.

LINKAGE CONVENTIONS

Work can be performed by Computing System/360 with the aid of functions provided by the System/360 Operating System. Some of these functions are the management of nested levels of control and action, as follows:

- Jobs are executed.
- Within a job, job steps are executed.
- Within a job step, a highest level task is created, and the program specified on the EXEC job control statement is given control. (Refer to the publication IBM System/360 Operating System: Job Control Language for details about the EXEC statement.)

When the first program of a job step is executed, the control level can change as a result of the following processes:

- The creation and termination of new tasks.
- The exchange of control between subprograms within the user's problem program.
- The exchange of control between the user's problem program and the control program.

Each of the above three processes is a linkage, and involves standard methods for giving control, passing data, and maintaining machine and program environments. These standard methods are the linkage conventions.

LINKAGE TERMINOLOGY

Linkages involve the use of subprograms or subtasks, and, therefore, each linkage results in an upward or downward change in control level (except when the XCTL macro-instruction is used; in this case, the control level does not change).

Linkage from a higher level program to a lower level program is called an entry linkage. It results in the giving of control to an entry point in the lower level program. Linkage from a lower level program to a higher level program is called an return linkage. It results in the giving of control to a return address in the higher level program.

An entry linkage is initiated in the higher level program by execution of a set of instructions referred to as a calling sequence. The linkage is completed by execution of entry code at the entry point of the lower level program. A return linkage occurs through execution of return code in the lower level program. This usually completes the linkage; that is, no linkage-associated instructions normally exist at the return address in the higher level program. (Calling sequence identifiers and instructions to interpret return codes are exceptions to this statement; they are discussed later.)

In addition to the giving and returning of control between two programs, it is usually necessary to communicate data, as follows:

- Implicit communication. Both programs know the location of the data because they were compiled, assembled, or linkage edited together.

- Explicit communication. At the time of the linkage, either the data or its location is passed between the programs. Communication by passing the data itself is called communication by value; communication by passing the location of the data is called communication by name.

Implicit communication can be used when the linkage is either a branch and link instruction or a CALL macro-instruction. However, if a program requiring data and using implicit communication is redesigned to run on a computer with a smaller main storage area, the program may have to be rewritten to use the LINK macro-instruction and explicit communication. If the program had originally used explicit communication, the changes required would be minor.

Data communicated explicitly is called a parameter. Parameters are further classified according to their use, as follows:

- Control program parameters, which are passed between the user's problem program and the control program when system macro-instructions are executed. These parameters are passed in either parameter registers or parameter lists.

- Problem program parameters, which are passed between subprograms of the user's problem program when a linkage occurs. These parameters are passed only in parameter lists.

LINKAGE TYPES

All linkages between subprograms of the user's problem program, and between the user's problem program and the control program, can be classified into four types:

- Type I - Direct Linkage: A branch and link instruction or a CALL macro-instruction is used to link two subprograms or the user's problem program and a problem state control program routine. (Most data management macro-instructions use this type of linkage at the interface between the user's problem program and the control program.)

- Type II - Supervisor-Assisted Linkage: A LINK, XCTL, or ATTACH macro-instruction is used to link two subprograms.

- Type III - Supervisor Linkage: A supervisor call (SVC) instruction is used to link to a control program routine that is executed in the supervisor state. (The SVC instruction is part of the macro-

expansion of a system macro-instruction, or it is in a control program routine entered by a direct linkage.)

- Type IV - Exit Linkage: A branch and link instruction or a load program status word (LPSW) instruction is used to enter a user's routine (called an exit routine) during execution of a system service in response to a macro-instruction; this linkage is called a synchronous exit. An LPSW is used to enter a user's routine when an asynchronous event occurs; this linkage is called an asynchronous exit.

The four linkage types serve as convenient descriptions of linkage interfaces that can exist. However, linkages can actually be more complex than the linkage types might indicate. Figure 1 shows one of the more complex cases.

In Figure 1, a LINK macro-instruction results in a supervisor-assisted (type II) linkage. The load module to be given control is module B. However, to give control to B, two linkages actually occur:

1. The first linkage is between load module A and the supervisor. The entry linkage uses an SVC instruction, and the return linkage uses an LPSW instruction.

2. The second linkage is between the supervisor and load module B. The entry linkage uses an LPSW instruction. The return linkage uses a RETURN macro-instruction, which results in a branch from module B to an SVC instruction in the supervisor. The branch instruction uses the contents of a return register, which the supervisor set with the address of the SVC instruction before giving control to module B.

The action of these linkages gives the effect of a direct linkage between load modules A and B.

A SAVE macro-instruction is shown at the entry point of module B, and a RETURN macro-instruction is shown at the location from which a return is made to the supervisor. These macro-instructions save and restore the contents of the registers used by module B.

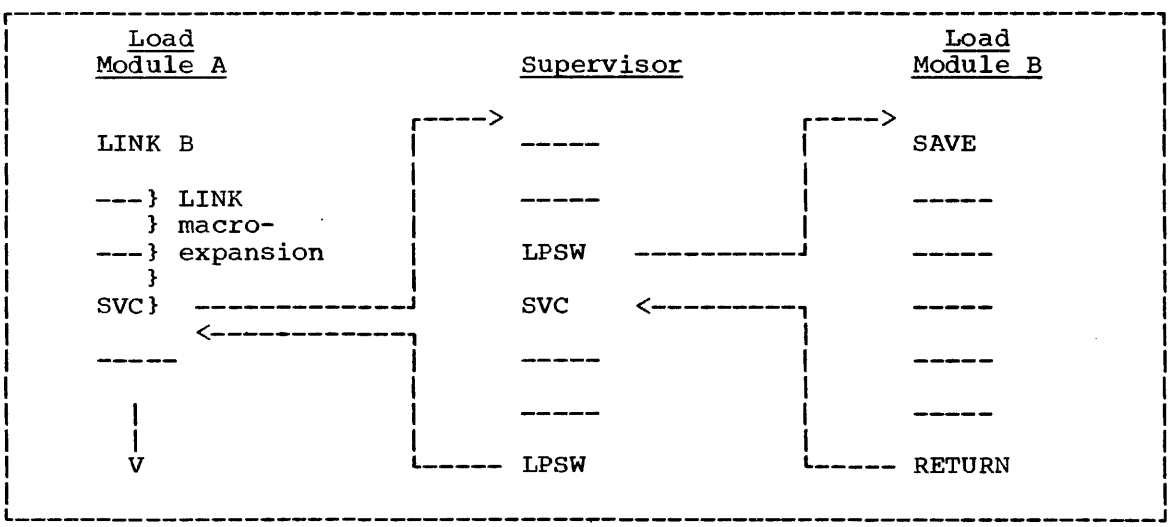


Figure 1. Linkages in LINK Macro-Instruction Execution

LINKAGE REGISTERS

The registers having specific roles in linkages are listed, and their functions described, in Table 2.

Table 2. Linkage Registers

Register Number	Register Name	Contents
0	Parameter register	Parameters to be passed to the control program.
1	Parameter register or Parameter list register	Parameters to be passed to the control program. Address of a parameter list to be passed to either the control program or a user's subprogram.
13	Save area register	Address of the register save area to be used by the called program.
14	Return register	Address of the location in the calling program to which control should be returned after execution of the called program.
15	Entry point register or Supervisor parameter list register or Return code register	Address of the entry point in the called program. Address of a parameter list to be used by the supervisor in a supervisor-assisted linkage. This list contains information needed by the supervisor to give control to the called program. A return code that indicates to the calling program whether or not an exceptional condition occurred during processing of the called program. The return code should be zero for a normal return or a multiple of four for various exceptional conditions.

Some of the linkage register identities and uses are shown in the following typical type I linkage calling sequence:

```
      CNOP    2,4
      LA      14,RET              load return address
      L       15,=V(SUBR)         load entry point address
      BALR    1,15                load parameter list address
      DC      A(PAR1,PAR2)        parameter list
RET   B       *+4(15)             branch, using return code
      B       NORMAL              branch if normal
      B       COND1               branch if condition 1
      B       COND2               branch if condition 2
      .
      .
      .
```

In the preceding sequence, a higher level program (the calling program) gives control to a lower level program (the called program) by branching to the address in register 15. Register 15 is the entry point register; it can be used to provide initial addressability in the called program.

Before branching, the calling program loads register 14, the return register, with the address to which the called program should return control.

Two parameters, PAR1 and PAR2, are passed to the called program, in a list pointed to by register 1, which is the parameter list register.

Before returning to the calling program, the called program may load register 15 with a return code. (In the above example, register 15 must be loaded with a return code.) In this use, register 15 is the return code register.

The return code should be 0 for a normal return. If the return code is a multiple of 4, it can be interpreted by a branch table in the calling program, as shown above. Another way of interpreting the return code is shown below:

```
RET   LTR     15,15               test return code
      BNZ     COND-4(15)          branch if not zero
```

Before the preceding calling sequence is executed, register 13, the save area register, must be loaded with the address of a save area that is provided by the calling program. The called program stores, in the save area, the contents of the registers that it will use.

If a supervisor call (type III) linkage results, parameters can be passed in a list as shown, or they can be passed in registers 0 and 1. In this use, registers 0 and 1 are parameter registers. Whether parameters are passed in a list or in registers depends on the type of the macro-instruction and on the number of parameters to be passed.

In a supervisor-assisted (type II) linkage, the called program need not be in main storage when the linkage occurs, and the calling program does not know its entry point address. The supervisor is given a symbolic program name by means of a supervisor parameter list pointed to by register 15. In this use, register 15 is the supervisor parameter list register. The supervisor then acquires the called program and loads its entry point address into register 15. Control appears to be given to the called program as in a type I linkage.

SAVE AREA USE

Registers that are not linkage registers must have their contents saved and restored by each lower level program that is given control by a higher level program. This conserves main storage, because the instructions to save and restore registers need not be in each calling sequence in the higher level program. With the exception of the ATTACH macro-instruction, the save area used is provided by the higher level program, and has a standard format so that all programs can save registers in a uniform manner. Save areas are chained together in ascending order so that register contents can be restored as control is returned to the higher level programs. Save areas can also optionally be chained together in descending order.

The SAVE macro-instruction has optional provisions for saving, in the save area, the entry point address and the return address associated with each linkage. The RETURN macro-instruction has an optional provision for marking the save area provided by the higher level program to indicate that the return has occurred.

These provisions can assist in the interpretation of program dumps taken by means of the test translator DUMP DATA and DUMP CHANGES macro-instructions.

Register Saving and Restoring Responsibilities

Every program, before it executes a type I, type II, or type III linkage, must provide a save area and place the address of this save area in register 13.[1] A program can use the same save area for all of its linkages; unless required for other purposes, register 13 need be loaded only once, when the program is entered. In the case of a reenterable program, the save area must be provided from a dynamically allocated area of storage.

The save area provided by the calling program is used by a called user's problem program in type I and type II linkages to save the contents of registers the called program will use. Register saving should be accomplished by using a SAVE macro-instruction. Because register saving should be the first action taken by the called program, the SAVE macro-instruction should be used at the entry point of the called program. The called program should use a RETURN macro-instruction to return control to the calling program, and to restore the saved registers from the save area.

The save area provided by the calling program is used by the control program in type III linkages. Before returning to the calling program, the control program may execute a type IV linkage to a user's synchronous exit routine. Although the contents of register 13 will be the same in both linkages, the exit routine must not attempt to use the save area provided by the calling program.

As in the case of the highest level program of a task, a routine entered by a type IV linkage can use any register (except the return register) without saving and restoring its contents. On execution of a RETURN macro-instruction, the control program restores register contents

---

[1]When the R forms of the GETMAIN and FREEMAIN macro-instructions are issued, a save area need not be provided, and the control program will not refer to or modify the contents of a main storage area pointed to by register 13.

automatically. To allow the use of standard linkage conventions, however, the control program provides a save area for use by highest level programs and by exit routines that are entered asynchronously on termination of a task (the ETXR and STAE routines) and on completion of a timer interval (the STIMER routine). This save area is located in subpool 0 of the job step and is pointed to by register 13.

Except for the SVC interruption, interruptions are not classified as linkages. When the control program processes any interruption, except for the SVC interruption that occurs in a type III linkage, it saves, in its own main storage area, the contents of all registers that it will use, and then restores these register contents before returning to the user's problem program.

Save Area

A save area occupies 18 full-words and is aligned on a full-word boundary. The save area words, their displacement in bytes from the area origin, and their contents are shown in Table 3.

Additional information on the contents of each of the words in a save area is given below:

- Word 1. An indicator byte followed by three bytes that contain the length of allocated storage. This field is used only by programs written in PL/I language.

- Word 2. The address of the save area used by the calling program. The address is passed to the calling program in register 13 by the next higher level program. The calling program must store the address in this word before it loads register 13 with the address of the current save area. This word contains all zeros if the current save area is provided by the supervisor for use by an asynchronous exit routine or the highest level program of a task.

- Word 3. The address of the save area provided by the called program, unless the called program is at the lowest level and does not have a save area. (The called program need have a save area only if it is itself a calling program, or if it executes a supervisor or data management macro-instruction other than SAVE, RETURN, or the R forms of GETMAIN and FREEMAIN.) If save areas are being chained together in descending order, the called program stores the save area address in this word. This word is not used by called control program routines.

- Word 4. The return address, which is in register 14 when control is given to the called program. The called program stores the return address in this word if it intends to modify register 14 or if the T operand is written in the SAVE macro-instruction. If the T operand is written in the RETURN macro-instruction, the called program changes the high-order byte of this word to all-ones just before it returns to the calling program. The all-ones byte indicates that the return occurred. None of these operations is performed by called control program routines.

- Word 5. The address of the entry point of the called program. This address is in register 15 when control is given to the called program. The called program stores the entry point address in this word if the T operand is written in the SAVE macro-instruction. This word is not used by called control program routines.

- Words 6 and 7. The contents of registers 0 and 1, respectively. The called program stores the register contents in these words if it so desires, or if the contents of registers 15 and 2 are saved. In the latter case, the SAVE macro-expansion contains a single STM instruction that also saves the contents of registers 0 and 1.

- Words 8 through 18. The contents of registers 2 through 12, in that order. The called program stores the register contents in these words if it intends to modify the registers.

Table 3. Save Area Words and Contents in Calling Programs

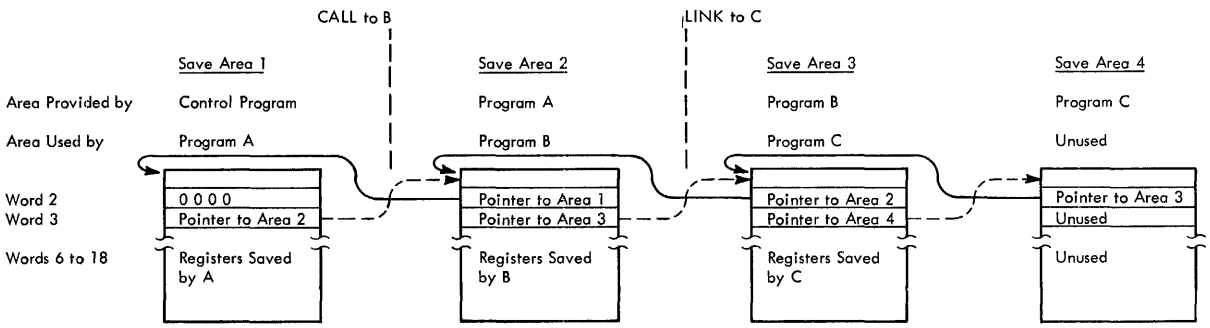
Word	Displacement	Contents
1	0	Indicator byte and storage length.
2	4	Address (stored by the calling program) of the save area used by the calling program. This save area is provided by the program that called the calling program.
3	8	Address (stored by the called program) of the save area provided by the called program.
4	12	Return address (register 14 contents - stored by the called program).
5	16	Entry point address (register 15 contents - stored by the called program).
6	20	Register 0
7	24	Register 1
8	28	Register 2
9	32	Register 3
10	36	Register 4
11	40	Register 5
12	44	Register 6
13	48	Register 7
14	52	Register 8
15	56	Register 9
16	60	Register 10
17	64	Register 11
18	68	Register 12

Save Area Chaining

The following examples illustrate the chaining of save areas when different linkages are used, and relate the chaining sequence to the concept of control levels. Each example concentrates on (1) the use of words two and three of a save area, (2) the contents of register 13 at the point of linkage, and (3) the responsibility of programs to provide save areas. Pointers to save areas in higher control level programs are shown as solid lines; pointers to save areas in lower control level programs are optional and are shown as dotted lines.

EXAMPLE 1: The job stream contains an EXEC statement for module ALPHA. ALPHA consists of program A and program B, which was included in the module as a result of a CALL macro-instruction. Program B contains a LINK macro-instruction to program C.

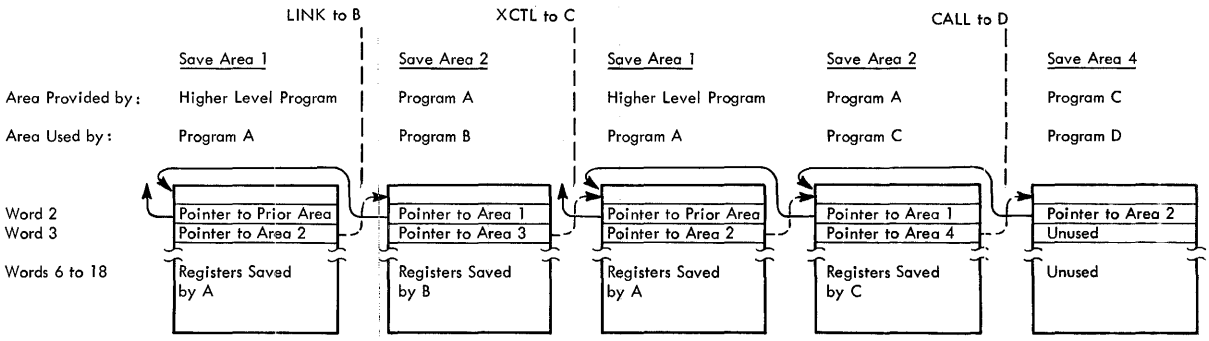
EXAMPLE 1



In this example, program A is considered to be at the highest control level and program C at the lowest. When program A receives control, word 2 of the save area provided by the control program contains zeros. At the time of each linkage, register 13 must point to the save area of the higher control level program. Since program C does not either contain a linkage to a lower control level or issue a system macro-instruction, save area 4 is not required. (Program C need only save register 13 until the return linkage.) The save area is shown here for generality, since program C might require the area during another execution.

EXAMPLE 2: Program A receives control from a higher level program and issues a LINK macro-instruction to program B, which in turn issues a XCTL macro-instruction to program C. Finally, program C calls program D. The major consideration here is the use of save area 2 by both program B (before the XCTL macro-instruction) and program C (after the XCTL macro-instruction).

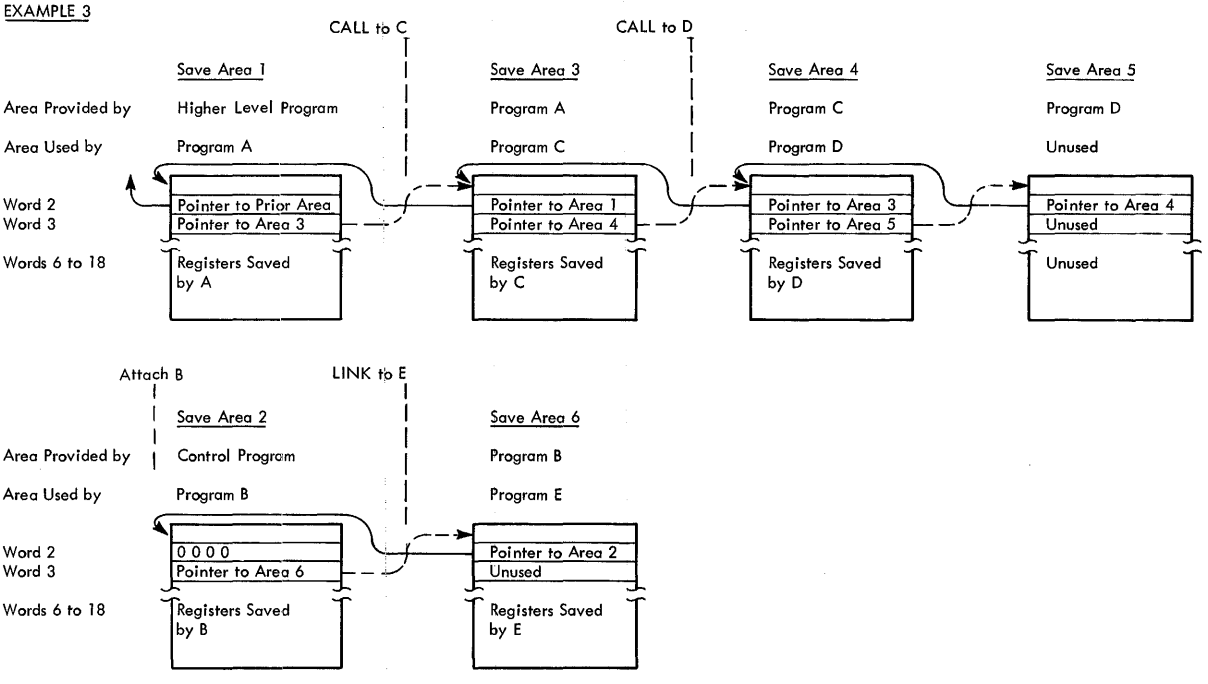
EXAMPLE 2



The XCTL macro-instruction has the same effect on the usage of save areas as that of a RETURN macro-instruction from program B followed by a LINK macro-instruction from program A to program C. Register 13 must point to save area 2 when the XCTL macro-instruction is executed. Program C then replaces program B as the lowest control level rather than introduce a new level of control. The linkage from program C to D introduces a lower control level. Note that program B provides a separate save area (area 3, not shown) that is not used in this execution.

EXAMPLE 3: Program A issues an ATTACH macro-instruction for a task that uses Program B. Program A then calls programs C and D. Program B links to program E.

EXAMPLE 3



Program A initiates a second sequence of control levels when it attaches a task. The first sequence of control levels starts with program A, the highest control level, and ends with program D, the lowest. The second sequence consists of program B at the highest level and program E at the lowest level. Also, the second sequence is at a lower control level than the first sequence even though the two operate in parallel. (It is understood that at some time program B will issue a RETURN macro-instruction and the task will be detached.)

CALLING SEQUENCE AND ENTRY POINT IDENTIFIERS

A calling sequence identifier is a 16-bit binary number in the second half-word of a full-word NOP instruction. The identifier can be specified by either the CALL or LINK macro-instruction. The NOP instruction is located at the return address if a CALL macro-instruction (or a hand-coded direct linkage) is used, or follows the SVC instruction if a LINK macro-instruction is used.

An entry identifier is a character string of up to 70 characters. It can be specified by the SAVE macro-instruction (described in "Supervisor Services").

LINKAGE INTERFACE RESPONSIBILITIES

There are three distinct linkage interfaces of which the programmer must be aware:

- The interface that a called program sees in type I, type II, and certain type IV linkages.
- The interface that a calling program sees in a type I linkage resulting from a hand-coded calling sequence.
- The interface that a calling program sees in a type I, type II, or type III linkage resulting from a supervisor or data management macro-instruction.

Conventions concerning exit routines (used in type IV linkages) are given in the descriptions of the macro-instructions used to invoke the exit routines.

Called Program Interface in Type I, Type II, and Certain Type IV Linkages

The conventions to be followed by a called program are independent of how the program is given control. That is, the called program need not be aware of whether it was entered through the use of CALL, LINK, XCTL, or ATTACH or through an asynchronous exit taken on termination of a task or completion of a timer interval. The called program is responsible for the following:

1. Saving the contents of registers 2 through 12 and 14 in the calling program's save area, if the called program modifies these registers, and subsequently restoring these registers before returning to the calling program.

2. Saving the contents of register 13 in the called program's save area, if the called program modifies this register, and subsequently restoring this register before returning to the calling program.

3. Ensuring that the program mask (PSW bits 36 through 39) and the program interruption control area (PICA) are the same upon exit from the called program as they were upon entry to it.

Item 1 can be accomplished by the SAVE and RETURN macro-instructions; items 2 and 3 must be accomplished by assembler language instructions.

The contents of the floating-point registers and the condition code (PSW bits 34 and 35) need not be the same upon exit from the called program as they were upon entry to it.

Calling Program Interface in a Type I Linkage Resulting From a Hand-Coded Calling Sequence

The calling program is responsible for the following:

1. Loading register 13 with the address of a save area.

2. Loading register 14 with the return address.

3. Loading register 15 with the entry point address.

4. Loading register 1, if necessary, with the address of a parameter list.

After execution of the calling sequence, the calling program can expect the following to occur as a result of execution of the remainder of the linkages:

1. The contents of registers 2 through 14, the program mask, and the program interruption control area will be unchanged.

2. The contents of registers 0, 1, and 15; the contents of the floating-point registers; and the condition code may be changed.

Calling Program Interface in Type I, Type II, and Type III Linkages Resulting From Supervisor and Data Management Macro-Instructions

The calling program is responsible for the following:

1. Ensuring that the entire macro-expansion and the literal pool currently being used are addressable by a base register other than registers 0, 1, 14, and 15.

2. Loading register 13 with the address of a save area.

3. If an XCTL macro-instruction is being executed, restoring the return register, and also the program mask, and the program interruption control area as they were upon entry to the calling program. The user can request that registers in the range 2 through 12 be restored. Register 13 must point to the save area in the program that called this calling program.

The calling program can expect the following to occur as a result of execution of the macro-instruction:

1. The contents of registers 2 through 13, the program mask, and the program interruption control area will be unchanged. Note that use of the SPIE macro-instruction results in an exception to the preceding statement: SPIE modifies the program mask and program interruption control area.

2. The contents of the floating-point registers will not be changed by a called control program routine, but they may be changed if the user's problem program is given control during the linkage (as in a type I or type II linkage, or if a synchronous exit routine is given control).

3. The contents of registers 0, 1, 14, and 15, and the condition code may be changed.

PASSING CONTROL INFORMATION TO A JOB STEP

The EXEC job control language statement can be used to pass control information to the first program of the specified job step. The control information is specified by characters written as the optional value of the PARM keyword operand. The following rules must be observed in writing the optional value:

- The optional value is delimited by: on the left, the equal sign of the keyword operand; and, on the right, a comma, if another operand follows, or a blank, if another operand does not follow.

- If the control information is to contain either a comma or a blank, the optional value must begin and end with a single quotation mark; these do not become part of the control information.

- If the optional value begins and ends with a single quotation mark, single quotation marks that are to be part of the control information must each be written in the optional value as a pair of single quotation marks. Note that the optional value cannot begin or end with an even number of single quotation marks.

- The control information cannot consist of more than 40 characters.

The control information is passed to the job step by means of a data area, a parameter list, and register 1. The data area consists of a half-word followed by the control information. The half-word contains a count of the number of control characters. The parameter list consists of a full-word that contains the address of the data area and has its high-order bit set to 1, giving the word the appearance of the last word in a variable-length parameter list. (Refer to the description of the VL operand of the CALL macro-instruction for a discussion of variable-length parameter lists.)

The control program places the data area and parameter list in a main storage area that it allocates from subpool zero of the job step. The data area and parameter list are aligned to half-word and full-word boundaries, respectively. The control program loads the address of the parameter list into register 1 and then gives control to the job step.

If the PARM field is omitted, the half-word count field in the data area is set to zero.

MACRO-INSTRUCTION DESCRIPTIONS

System macro-instructions are presented in this publication by means of macro-instruction descriptions, each of which is organized in accordance with the following outline:

1. Title - the mnemonic operation code of the macro-instruction; a phrase explaining either the meaning of the mnemonic or the function of the macro-instruction; and, where applicable, a parenthesized letter stating the type of the macro-instruction (R or S).

2. Function - a brief summary of the services provided.

3. Format Description - an illustration showing how and when operands are to be written.

4. Operand Descriptions - detailed information about writing each operand, including any cautions applicable to a particular operand.

5. Execution - reference material describing the normal use or execution of the macro-instruction.

6. CAUTIONS - warnings of any special restrictions on the use of the macro-instruction. In some cases, the results of improper use are described.

7. EXCEPTIONAL RETURNS - material describing return codes and synchronous and asynchronous exit routines. (Refer to "Linkage Conventions.")

8. ENVIRONMENT - description of the use of the macro-instruction with a subset of the control program and a description of the services available.

9. EXAMPLES - one or more specific examples showing how the macro-instruction is written and what it does.

10. PROGRAMMING NOTES - tutorial material describing the use of the macro-instruction and the services that it requests.

11. L- AND E-FORM USE - a statement of the abnormal operand requirements, if any, of the L and E forms of an S-type macro-instruction.

Items 1 through 4 are included in all macro-instruction descriptions. The remaining outline items are used only as appropriate. When items 6 through 11 are included, they are identified by the indicated all-caps headings.

SECTION 2: SUPERVISOR SERVICES

The supervisor provides a variety of services that help the user manage programs and tasks, handle exceptional conditions, operate the interval timer, and write to the operator or log. The supervisor's program management facility enables operation of simple, overlay, and dynamic programs. The user requests supervisor services through the macro-instructions described in this section.

For ease of reference, these macro-instructions are grouped in subsections according to functions. The order of the subsections, the macro-instructions covered under each, and the general function of each group are as follows:

- Simple Program Management: CALL, SAVE, and RETURN. These macro-instructions provide standard linkage between routines to form them into simple programs. SAVE and RETURN are also applicable to the dynamic program management function (described below).
- Overlay Program Management: SEGLD and SEGWT. The first macro-instruction provides overlap between segment loading and processing while the second delays processing until the requested segment is in main storage.
- Dynamic Program Management: LINK, XCTL, LOAD, DELETE, and IDENTIFY. These macro-instructions provide supervisor-assisted linkages between load modules to form a program dynamically during its execution.
- Main Storage Management: GETMAIN and FREEMAIN. These macro-instructions dynamically allocate storage to a task and return allocated storage to the control program.
- Task Creation and Management: ATTACH, DETACH, CHAP, and EXTRACT. These macro-instructions create tasks and remove them from the system, and provide the basic means for task management.
- Task Synchronization: WAIT, WAITR, POST, ENQ, and DEQ. These macro-instructions enable a task to synchronize itself with another task, or with a control program service.
- Exceptional Condition Handling: SPIE, STAE, ABEND, and CHKPT. These macro-instructions provide for program interruptions, abnormal terminations, and checkpoints.
- General Services: TIME, STIMER, TTIMER, WTO, WTOR, and WTL. These macro-instructions enable a program to set, check, and cancel a time interval and to write to the log and to the operator (with or without a reply.)

Some supervisor macro-instructions request services that are affected by the control program options that can be excluded by an installation. The control program options and storage requirements are discussed in detail in the publication, IBM System/360 Operating System: Storage Estimates, Form C28-6551.

The operating system from which all control program options have been excluded is referred to as the primary control program. It provides for stacked job processing with sequential scheduling (jobs are processed as they are provided as input to the system) and for single task operation. The function and performance of the primary control program can be increased by inclusion of the following options:

Option 1: Multiple wait

Option 2: Multiprogramming with a fixed number of tasks

Option 3: Identify

Option 4: Multiprogramming with a variable number of tasks

A. Scheduling single job with work queue directory in main storage

B. Scheduling single job with work queue directory on direct-access storage

C. Scheduling multiple jobs with work queue directory in main storage

D. Scheduling multiple jobs with work queue directory on direct-access storage

Option 5: Additional transient areas and control

Option 6: Timing

A. Time

B. Interval timing

Option 7: Alternate console

Option 8: Composite console

Option 9: Protection

Option 10: Priority scheduling

Option 11: Input readers/interpreters

Option 12: Output writers

Option 13: Job step timing

Option 14: Rollout/rollin

Each installation's guide should be consulted to determine which control program options have been excluded from the system. The manner in which individual services are affected by the inclusion or exclusion of particular options is discussed in detail in each macro-instruction description. Table 4 summarizes the services so affected.

Table 4. Services Affected by Including or Excluding Control Program Options

Macro-Instruction	Option Included	Option Excluded	Result
ABEND		4	The entire job step is terminated abnormally
ATTACH		4	Refer to the Environment discussion in the macro-instruction
CHAP		4	NOP
CHKPT	4		NOP
DEQ		4	NOP
DETACH		4	NOP
ENQ		4	NOP
EXTRACT		4	Only TIOT address is provided
FREEMAIN		4	Subpool ignored List request invalid
GETMAIN		4	Subpool ignored List request invalid
IDENTIFY		3 & 4	Refer to the Environment discussion in the macro-instruction
	3	4	
SEGLD		4	NOP
SPIE		4	The exit routine applies to the job step
STAE		4	NOP
STIMER		6B	NOP
TIME		6A & 6B	Only the date is provided
TTIMER		6B	NOP
WAIT		1,2, & 4	Meaningful for only one event
WAITR		2	Treated as a WAIT
WTL		12	NOP

CALL -- Call a Program (S)

The CALL macro-instruction passes control from a program, load module, or segment of an overlay load module (each called a program for convenience) to a specified entry point in another program. The program issuing the CALL macro-instruction is referred to as the calling program; the program receiving control is referred to as the called program. Except when the overlay supervisor can be used, the called program must be in main storage when the CALL macro-instruction is executed. The called program is brought into main storage in one of two ways:

1. As part of the load module issuing the CALL. In this case, the CALL macro-instruction must specify an entry point. When the linkage editor processes a load module containing such a CALL, it includes the called program in the load module.

2. As the load module specified by a LOAD macro-instruction. In this case, the CALL macro-instruction must specify the program to be called by indicating that the address of its entry point will be loaded into register 15 (the entry point register) before execution of the CALL macro-instruction. The LOAD macro-instruction must precede the first CALL for the program.

The called program returns control to the calling program by issuing a RETURN macro-instruction.

Name	Operation	Operand
[symbol]	CALL	{entry-symbol / (15)}[,({param-addr,}...)[,VL]] [,ID=absexp]

entry
specifies the entry point to which control is to be passed. If the symbolic name of an entry point is written, a V-type address constant is generated as part of the macro-expansion; control is given to the called program by a branch to the address in register 15 (the entry point register).

If (15) is written, the actual address of the entry point must have been loaded into register 15 before execution of this macro-instruction.

param
specifies an address to be passed as a parameter to the called program. The param operands must be written in a sublist, as shown in the format description. If one or more param operands are written, a problem program parameter list is generated. It consists of a full-word for each operand. Each full-word is aligned on a full-word boundary and contains, in its three low-order bytes, the address to be passed. The addresses appear in the parameter list in the same order as in the macro-instruction.

When the called program is entered, register 1 (the parameter list register) contains the address of the problem program parameter list.

If the param operand is omitted in a standard form of the macro-instruction, register 1 is not set to zero.

VL

specifies that the sign bit is to be set to 1 in the last full-word in the problem program parameter list.

The parameter list has a fixed length if it is to contain a certain, known number of parameters every time the called program is given control. The list has a variable length if it can contain a varying number of parameters. Only in the latter case should the VL operand be written in order to mark the end of the list.

If the list has a variable length and if register notation is used to write the last param address, the user's problem program can set the sign bit in the designated register to 1. If this is done, the VL operand need not be written.

ID

specifies a binary calling sequence identifier. The maximum value of the identifier is $2^{16}-1$. When this operand is written, a full-word NOP instruction appears at the end of the macro-expansion. The NOP instruction contains the operand value in its two low-order bytes.

Upon entry to and return from the called program, register 14 (the return register) contains one of the following:

- If the ID operand was written, register 14 contains the address of the last instruction (the NOP instruction) in the macro-expansion.
- If the ID operand was omitted, register 14 contains the address of the first byte following the macro-expansion.

CAUTIONS: The called program operates at the same control level as the caller. If the called program issues an XCTL macro-instruction, the caller cannot expect to regain control.

If the entry operand is written as a symbolic name, a V-type address constant is generated by the assembler, and the linkage editor can make the called program part of the calling program's load module as part of the automatic library call procedure. The symbolic name must be either the name of a control section or an assembler language ENTRY statement operand in the called program.

If the entry operand is written as (15), a V-type address constant is not generated. If the called program is not part of the calling program's load module, a LOAD macro-instruction must be executed (to bring the program to be called into storage) before the CALL macro-instruction is issued.

The supervisor has no control over entry to a program by means of the CALL macro-instruction. Therefore, when a serially reusable program can be entered by two or more tasks using only CALL macro-instructions or a combination of CALL macro-instructions and supervisor-assisted linkages (LINK, XCTL, and ATTACH), the ENQ and DEQ macro-instructions must be used, to ensure that only one task at a time uses the called program. (Refer to "Task Synchronization" for information on the use of ENQ and DEQ.)

EXCEPTIONAL RETURNS: The called program can specify a return code in the RETURN macro-instruction. When the RETURN macro-instruction is executed, the return code is loaded into register 15 (the return code

register). When the calling program resumes execution, it can interrogate the return code in register 15.

EXAMPLES: In the following examples, EX1 gives control to an entry point named ENT and specifies a calling sequence identifier of 2. No parameters are passed.

EX2 gives control to an entry point whose address is contained in register 15. Two parameters, ABC and DEF, are passed. Because the parameter list has a variable length, the VL operand is specified. No calling sequence identifier is specified.

```
EX1     CALL  ENT,ID=2
EX2     CALL  (15),(ABC,DEF),VL
```

PROGRAMMING NOTES: If register notation is used to write any param operands, instructions to store the contents of the designated registers in the parameter list are the first executable instructions of the macro-expansion. The first of these instructions can be referred to by the symbol (if any) in the name field of the macro-instruction.

If the ID operand is written, a NOP instruction follows the parameter list. The return address is the same with or without the ID operand.

When the CALL macro-instruction is executed, it gives control to the called program by branching to the address in register 15.

The (15) entry operand and LOAD macro-instruction combination is most useful when the same program is to be called many times during execution of the calling program, but is not needed in main storage throughout execution of the calling program. If the LINK macro-instruction is used instead of LOAD and CALL, more execution time may be required because the supervisor may search main and external storage for the program each time the LINK is issued. If the CALL macro-instruction is used and a symbolic name written for the entry operand, the called program resides in storage throughout execution of the calling program. This wastes main storage if the called program is not needed during all of the calling program's execution.

L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B except for the following special operand requirements:

Operand	L Form	E Form
entry	not allowed	required
VL	allowed	allowed only if the param operand that results in the last address in the parameter list is also written in the macro-instruction
ID	not allowed	allowed
param	required	allowed

Only the param sublist operand and the VL operand can be written in the L form of the macro-instruction. The param operand must be preceded by a comma, as shown in the macro-instruction format.

All operands can be written in the E form of the macro-instruction. If any param operands are written, the addresses are stored in the remote parameter list in accordance with their positions in the sublist. For example, if the sublist is (A,,B), addresses A and B are stored in the first and third words of the parameter list.

If the remote parameter list is of variable length, the VL operand in the E-form macro-instruction should be written only if the param operand corresponding to the last full-word in the list is also written.

If the entry operand is (15), register 15 is not used as a working register in the macro-expansion; addresses are formed and placed in the remote list using only register 14.

SAVE -- Save Register Contents

The SAVE macro-instruction is written at the entry point of a program. Upon entry to the program, SAVE stores the contents of specified registers in a save area provided by the program from which control was given. The saved register contents are reloaded by execution of a RETURN macro-instruction.

The SAVE macro-instruction can also generate an entry-point-identifier character string.

Name	Operation	Operand
[symbol]	SAVE	(reg_1-integer[,reg_2-integer]),[T] [,id-{characters / *}]

reg_1,reg_2
specifies the range of registers to be stored in the save area of the calling program. (This area is pointed to by register 13; refer to "Linkage Conventions" in Section 1.) The operands are written as decimal numbers. They should be so written that, when inserted in a STM instruction, they cause desired registers in the range of 14 through 12 (14, 15, 0 through 12) to be stored. Registers 14 and 15, if specified, are saved in words 4 and 5 of the save area. Registers 0 through 12, if specified, are saved in words 6 through 18 of the save area. The contents of a given register are always saved in a particular word in the save area. For example, register 3 is always saved in word 9 of the save area, even if register 2 is not saved.

If reg_2 is omitted, only the register specified by reg_1 is saved.

T
specifies that, if not saved by the first operand, registers 14 and 15 are to be saved in words 4 and 5 of the save area. If the T and reg_2 operands are present and the reg_1 operand is 14, 15, 0, 1, or 2, all registers from 14 through the reg_2 value are saved.

id
specifies the identifier of the entry point at which the SAVE macro-instruction is located. The operand is a character string and can consist of up to 70 characters. Because it can have a length greater than eight characters, it can be a combination of a data set name and a program name, or some other complex name.

If this operand is written as an asterisk, the entry point identifier is the same as the symbol in the name field of the macro-instruction; if the name field is blank, the entry point identifier is assumed to be the name of the control section containing the macro-instruction.

CAUTIONS: A SAVE macro-instruction must not be used at the beginning of an exit routine except the ETXR, STAE, and STIMER exit routines. No save area is provided in other asynchronous exits, and the save area pointed to in synchronous exits must not be used.

An exit routine may use any register (except register 14) without saving and restoring its contents. The control program saves the required registers before giving control to the exit routine, and, on execution of a RETURN macro-instruction, restores register contents automatically.

EXAMPLES: In the following examples, EX1 saves registers 14 through 10. Registers 14 and 15 (and 0 and 1, incidentally) are saved because the T operand is written. The entry point identifier is F4RTNA7B99. EX2 saves registers 3 and 4. The entry point identifier is EX2.

```
EX1     SAVE  (2,10),T,F4RTNA7B99
EX2     SAVE  (3,4),,*
```

PROGRAMMING NOTES: The SAVE macro-instruction is expanded as follows:

- A branch to the next executable instruction.
- A one-byte count field for the number of characters in the entry point identifier.
- The entry point identifier.
- An alignment byte (if one is necessary).
- The next executable instruction (a STM instruction).

When the T and reg_2 operands are present and the reg_1 operand is 14, 15, 0, 1, or 2, a single STM instruction is generated to store registers 14 through the reg_2 value. When the reg_1 value is 3 to 12, two STM instructions are generated: one stores the contents of registers 14 and 15; the other stores the contents of the registers from the reg_1 value through the reg_2 value.

A symbol in the name field of a SAVE macro-instruction is an entry point name. The entry point name and the entry point identifier are the same only if the last operand of the macro-instruction is an asterisk. The entry point name is used in passing control to the entry point. If a program in another object module is to branch to the entry point, the entry point name should be an operand of an ENTRY assembler language statement provided in the current object module by the programmer. If no symbol is written in the name field of the macro-instruction and an asterisk is written as the id operand, the entry point identifier is the name of the control section in which the macro-instruction appears. A program in another object module can branch to this entry point name.

Because a register's contents are always saved in a particular word in a save area, the programmer can partially interpret the save area's contents in a main storage dump without knowing which registers were saved.

RETURN -- Return to a Program

The RETURN macro-instruction indicates normal termination and returns control to a higher level program or task, or to the control program. This macro-instruction's exact function depends on where it is used:

1. In the highest level program of a subtask, the RETURN macro-instruction indicates that the subtask is complete. It terminates the subtask and, optionally, notifies the next higher level task of the subtask's completion.

2. In the highest level program of the highest level task of the job step, the RETURN macro-instruction indicates that the task and job step are complete. It terminates the step, and returns control to the job scheduler.

3. In other than the highest level program of a task, the RETURN macro-instruction indicates that the program is complete. It terminates the program and returns control to the next higher level program. The program receiving control can be one of the following:

   a. The program that issued a CALL or LINK macro-instruction to give control to the program containing the RETURN.
   b. The program that issued a LINK macro-instruction to give control to a program that, in turn, issued an XCTL macro-instruction to give control to the program containing the RETURN.

4. In a synchronous exit routine, the RETURN macro-instruction indicates that the routine is complete. It terminates the routine and returns control to the control program.

5. In an asynchronous exit routine, the RETURN macro-instruction indicates that the routine is complete. It terminates the routine and returns control to the control program which returns to the program that was interrupted to allow execution of the exit routine.

The RETURN macro-instruction can reload the registers whose contents were saved by execution of a SAVE macro-instruction.

Name	Operation	Operand
[symbol]	RETURN	[(reg_1-integer[,reg_2-integer])][,T] [,RC={absexp / (15)}]

reg_1,reg_2
specifies the range of registers to be reloaded from the save area of the program receiving control. The operands are written as decimal numbers. They should be so written that, when inserted in a LM instruction, they cause the loading of registers in the range from 14 through 12 (14, 15, 0 through 12). Registers 14 and 15, if specified, are restored from words 4 and 5 of the save area. Registers 0 through 12, if specified, are restored from words 6 through 18 of the save area. If reg_2 is omitted, only the register specified by reg_1 is restored. If both reg_1 and reg_2 are omitted, no registers are restored.

The address of the save area must have been loaded into register 13 before execution of this macro-instruction.

T

specifies that a byte containing all ones is to be moved to the high-order byte of word 4 in the save area. This action occurs after completion of the register reloading specified by the first operand. The all-ones byte indicates that the return occurred.

RC

specifies a return code that is to be placed in the 12 low-order bits of register 15 (the return code register). The value of the absolute expression should be a multiple of 4 in the range from 0 through 4092.

If (15) is written, the return code must have been loaded into register 15 before execution of this macro-instruction.

If this operand is omitted, register 15 is loaded as specified by the reg_1 and reg_2 operand values.

This operand has no effect if the macro-instruction is executed by an asynchronous exit routine. The control program, upon receiving control, replaces the return code in register 15 with the original contents of the register.

CAUTIONS: A BR 14 instruction is always the last instruction in the RETURN macro-expansion. Register 14 (the return register) must be restored by means of the first operand of the macro-instruction; or, it must be correctly loaded before the macro-instruction is executed.

The RETURN macro-instruction can be used to terminate a synchronous exit routine. In this case, the first and second operands of the macro-instruction should not be written; the RETURN macro-instruction results in only a BR 14 instruction and may optionally load a return code.

A RETURN macro-instruction with no operands can be used to return from an asynchronous exit routine; it results in only a BR 14 instruction.

The control program saves registers 2 through 14 before giving control to a synchronous exit routine; it saves registers 14 through 2 (14, 15, 0, 1, and 2) before giving control to a SPIE routine, and saves all registers before giving control to any other asynchronous exit routine. The control program also reloads these registers when a return is made from the exit routine.

A RETURN macro-instruction should not be issued by the highest level program of a task that has incomplete subtasks; if it is, the task and all its incomplete subtasks are terminated abnormally. Also, a RETURN macro-instruction should not be issued by a program that includes exit routines whose execution may be required at a later time; if issued, the RETURN macro-instruction may cause deletion of the program and abnormal termination on a subsequent type IV linkage.

There are certain restrictions on the use of the RETURN macro-instruction in an overlay program; for details, refer to "Overlay Program Management."

EXAMPLES: In the following examples, EX1 is a RETURN macro-instruction that restores registers 2 through 10. All ones are placed in the high-order byte of word 4 in the save area. EX2 restores registers 14 through 5 and places a return code of 12 in register 15. EX3 is for

termination of a synchronous exit routine. A return code should have been loaded into register 15 by the user's problem program.

```
EX1     RETURN   (2,10),T
EX2     RETURN   (14,5),RC=12
EX3     RETURN   RC=(15)
```

PROGRAMMING NOTES: When issued by the highest level program of the highest level task of a job step, the RETURN macro-instruction indicates that the job step is finished. Control is given to the job scheduler. The job scheduler compares the return code with the condition code parameter of the appropriate JOB or EXEC control statement, to determine whether or not a subsequent job step should be executed. (Refer to the publication IBM System/360 Operating System: Job Control Language for an explanation of these control statements.)

When a RETURN macro-instruction terminates a task other than the highest level task of a job step, the return code is placed in the task control block (TCB) of the task issuing the RETURN. The return code is stored in the task completion code field of the task control block, where it can be interrogated by the next higher level task or the STAE routine of the terminating task. The STAE routine is entered only if the RETURN causes the task to be terminated abnormally. (The STAE routine is specified by the STAE macro-instruction, which is described in "Exceptional Condition Handling.")

If the terminating task was created by means of an ATTACH macro-instruction having an ECB operand, the specified event control block is posted, and the return code is stored in the task control block and in bits 2 through 31 of the POST code field of the event control block.

When a RETURN macro-instruction terminates a program other than an exit routine or highest level program in a task, the return code can be interrogated in register 15 by the next higher level program.

OVERLAY PROGRAM MANAGEMENT

The programmer can organize his program in an overlay structure by dividing it into segments according to the functional relationships of control sections. Two or more segments that can be loaded at different times can be assigned the same storage addresses by the linkage editor. The publication IBM System/360 Operating System: Linkage Editor contains a detailed discussion of the designing and structuring of an overlay program and of communication between segments of such a program.

The programmer uses linkage editor control statements to specify the relationship of segments within the overlay structure. The segments of the program (load module) are placed in a library so that the control program can load them separately when the program is executed. However, the programmer must be aware of how his program can communicate with the control program during execution. There are four ways in which he can have his program request the use of the overlay facilities.

1. By a CALL macro-instruction, which gives control to a symbol defined in another segment. The segment is loaded by the control program, if necessary.

2. By a branch instruction, which gives control to a symbol defined in another segment. The segment is loaded by the control program, if necessary.

3. By a SEGLD macro-instruction, which requests loading of a segment. Processing continues in the requesting segment while the requested segment is being loaded.

4. By a SEGWT macro-instruction, which requests loading of a segment and stops processing in the requesting segment until the requested segment is in main storage. After a SEGLD macro-instruction, a SEGWT macro-instruction, specifying a control section or entry name in the segment requested by the SEGLD, can be issued to stop processing in the requesting segment until the requested segment is in main storage.

PROGRAMMING NOTE: If exit routines are used (e.g., a timer exit), they should be placed in the root segment.

CALL Macro-Instruction in Overlay Management

The CALL macro-instruction refers to an external name that is an entry point of the segment to which control is to be passed. The external name is the name of a control section in the requested segment, or it is defined by an assembler language ENTRY statement in the requested segment. The requested segment and any segments in its path are loaded if they are not part of a path already in main storage. After the requested segment has been loaded, control is given to it.

Branch Instruction in Overlay Management

Any of the instruction sequences shown in Figure 2 can be used in place of the CALL macro-instruction to request loading and branching to a segment. In these instructions, 15 is a register into which is loaded a four-byte V-type address constant that is the address of an entry name or control section name defined in the requested segment. R_1 can be any other register but is usually register 14.

As a result of using any of the branch instructions listed in Figure 2, the requested segment and any segments in its path are loaded if they are not part of a path already in main storage. Control is then given to the requested segment at the location specified by the address constant V(NAME).

Examples 5, 6, and 7 in Figure 2 are unconditional branches. Branches on other conditions are also allowed.

If format $D_2(X_2,B_2)$ is used, the base register or index register can be loaded with the address constant. The remaining two fields must be zero.

If format $S_2(X_2)$ is used, the index register must be loaded with the address constant, and the base address and displacement must both be zero.

Example	Name	Operation	Operand
1		L BALR	15,=V(NAME) R_1,15
2	 ADCON	L BALR . . . DC	15,ADCON R_1,15 V(NAME)
3		L BAL	15,=V(NAME) R_1,0(0,15)
4		L BAL	15,=V(NAME) R_1,0(15)
5		L BCR	15,=V(NAME) 15,15
6		L BC	15,=V(NAME) 15,0(0,15)
7		L BC	15,=V(NAME) 15,0(15)

Figure 2. Branching Instructions

CAUTION: The address constant loaded in register 15 must be a four-byte V-type address constant. The high-order byte is reserved for use by the control program, and must not be altered by the user's problem program.

Inclusive Branches: A branch instruction between inclusive segments is always valid. A return from the requested segment can be made by means of the address stored in R_1 by the BAL or BALR instruction.

Exclusive Branches: A branch instruction between exclusive segments is valid only if a common segment also contains a V-type address constant that refers to the external symbol being branched to. The external symbol must satisfy all of the following conditions; it must be:

- Referred to by a V-type address constant in the requesting segment.
- Defined as an entry name or control section name in the requested segment.
- Referred to by a V-type address constant in a common segment.

A return from the requested segment can be made only by use of another exclusive branch instruction. It cannot be made as in an inclusive branch.

SEGLD -- Load Segment While Processing (R)

The SEGLD macro-instruction causes a specified segment to be loaded into main storage while the segment issuing the macro-instruction continues to be processed.

Name	Operation	Operand
[symbol]	SEGLD	externalname-symbol

externalname
specifies the name of a control section or an entry name in the requested segment. The macro-expansion results in a V-type address constant, and therefore the external name need not be identified by an EXTRN statement.

CAUTION: An exclusive reference should not be used in a SEGLD macro-instruction.

ENVIRONMENT: If option 4 has been excluded from the system, the SEGLD macro-instruction will be treated as a NOP at execution.

EXAMPLES: In both of the following examples, a SEGLD macro-instruction causes loading of the requested segment and any segment in its path, if they are not part of a path already in main storage. Processing resumes at the next sequential instruction while the segment or segments are being loaded. Following execution of EX1, control is given to the requested segment by the CALL macro-instruction; following execution of EX2, control is given by a branch instruction. Processing is stopped upon execution of the CALL or branch until loading of the requested segment is complete.

```
EX1     SEGLD    NAME
          .
          .
          .
        CALL     NAME

EX2     SEGLD    NAME
          .
          .
          .
        L        15,=V(NAME)
        BCR      15,15
```

SEGWT -- Load Segment Before Further Processing (R)

The SEGWT macro-instruction causes a specified segment to be loaded into main storage. Processing of the segment issuing the macro-instruction is stopped until the requested segment is loaded.

Name	Operation	Operand
[symbol]	SEGWT	externalname-symbol

externalname
specifies the name of a control section or an entry name in the requested segment. The macro-expansion results in a V-type address constant, and therefore the external name need not be identified by an EXTRN statement.

CAUTION: An exclusive reference should not be used in a SEGWT macro-instruction.

EXAMPLES: In the following examples, the SEGWT macro-instruction ensures that no further processing will take place until the requested segment and all segments in its path are loaded (if they are not already in main storage). Control is returned to the next sequential instruction in the requesting segment.

In EX1, the SEGLD macro-instruction causes overlap between processing and segment loading. The SEGWT macro-instruction prevents further processing in the requesting segment until the segment containing the data at DATA is in main storage. If the requested segment is in main storage, control is immediately returned to the instruction that follows the SEGWT macro-instruction.

In EX2, no overlap is provided. The SEGWT macro-instruction initiates loading. The task issuing the macro-instruction is placed in a wait condition until the requested segment is in main storage.

```
EX1     SEGLD    NAME
          .
          .
          .
        SEGWT    NAME
        L        R1,ADCON
          .
          .
          .
ADCON   DC       A(DATA)

EX2     SEGWT    NAME
        L        R1,=A(DATA)
```

PROGRAMMING NOTES: If the contents of a main storage location in the requested segment are to be processed, the name of the location must be referred to by an A-type address constant.

DYNAMIC PROGRAM MANAGEMENT

LINK -- Link to a Load Module (S)

The LINK macro-instruction gives control from one load module to an entry point in another specified load module. If the specified load module is reenterable and a copy is in main storage, it is used. If the load module is serially reusable, and if a copy is in main storage and not being used (in a type II linkage), it is used; if this copy is being used, the request for its use is queued. If the load module is not reusable and an unused copy is in main storage, it is used; if this copy has been used (in a type-II linkage), a new copy is loaded. If no copy of the load module is in main storage, a copy is loaded.

The linkage relationship between the load modules that give and receive control is the same as it would be if a CALL macro-instruction were used instead of the LINK macro-instruction. The module to be given control will execute at a lower control level than the module issuing the LINK. The lower level module can return control to the higher level module by executing a RETURN macro-instruction.

Name	Operation	Operand
[symbol]	LINK	{EP=symbol / EPLOC=addr / DE=addr}[,DCB=addr][,PARAM=({addr,}...) [,VL=1]][,ID=absexp]

EP
specifies the symbolic name of an entry point in the load module to be given control.

The entry point name must either be a name contained in the directory of a partitioned data set (member name or alias) or have been identified to the control program through the use of the IDENTIFY macro-instruction.

EPLOC
specifies the address of a double-word that contains the symbolic name of an entry point in the load module to be given control. The name must be left-justified in the double-word, and, if the name is less than eight characters, the double-word must be filled out with trailing blanks. The double-word can be aligned on a byte boundary.

DE
specifies the address of the name field of a list entry describing the load module to be given control. The entry contains information previously extracted from the directory of a partitioned data set by a BLDL macro-instruction. (Refer to "Basic Partitioned Access Method" in Section 3 for a description of the BLDL macro-instruction.)

If the DE operand is written, the DCB operand must be identical to the DCB operand specified in the corresponding BLDL macro-instruction.

DCB
specifies the address of a data control block opened for a private library (partitioned data set) that is to be searched for the load module to which control is to be given. If the EP or EPLOC operand was written and the load module is not found in the library specified by the DCB operand, the link library is searched.

If the DCB operand is omitted, the load module is assumed to be in either the job library or the link library. The job library, if one exists, is searched first.

The data control block addressed by this operand must specify use of the EXCP macro-instruction, and must have been opened for INPUT before execution of the LINK macro-instruction. Refer to the publication IBM System/360 Operating System: System Programmer's Guide, Form C28-6550 for a description of the EXCP macro-instruction. This data control block must not be used for any purpose other than the LINK, XCTL, LOAD, ATTACH, and BLDL macro-instructions. The data control blocks for the job library and link library are always open.

PARAM
specifies, as a sublist, address parameters to be passed from the load module issuing the macro-instruction to the load module to be given control. If one or more operands are written in the sublist,

a problem program parameter list is generated. It consists of a full-word for each operand. Each full-word is aligned on a full-word boundary and contains, in its three low-order bytes, the address to be passed. The addresses appear in the parameter list in the same order as in the macro-instruction.

When the load module to be given control is entered, register 1 (the parameter list register) contains the address of the problem program parameter list.

If the PARAM operand is omitted, register 1 is not set to zero.

VL

specifies that the sign bit is to be set to 1 in the last full-word in the problem program parameter list.

The parameter list has a fixed length if it is to contain a certain, known number of parameters every time the lower level load module is given control. The list has a variable length if it can contain a varying number of parameters. Only in the latter case should the VL operand be written in order to mark the end of the list.

If the list has a variable length and if register notation is used to write the last PARAM address, the user's problem program can set the sign bit in the designated register to 1. If this is done, the VL operand need not be written.

ID

specifies a binary calling sequence identifier. The maximum value of the identifier is $2^{16}-1$. If this operand is written, a full-word NOP instruction appears at the end of the macro-expansion (after the SVC instruction). The NOP instruction contains the operand value in its two low-order bytes.

CAUTION: The supervisor will abnormally terminate the task issuing the LINK if the specified load module cannot be located.

If the "only loadable" (OL) attribute was specified when the load module was processed by the linkage editor, an attempt to link to the module will cause abnormal termination.

EXCEPTIONAL RETURNS: The load module entered by a LINK can specify a return code in the RETURN macro-instruction. When the RETURN macro-instruction is executed, the return code is loaded into register 15 (the return code register). When the load module that issued the LINK resumes execution, it can interrogate the return code in register 15.

ENVIRONMENT: The following apply if option 4 was excluded from the system:

- An entry point identified to the control program in an IDENTIFY macro-instruction cannot be specified in a LINK macro-instruction.
- A LINK macro-instruction specifying a load module that was not previously brought into main storage by a LOAD macro-instruction usually will load the specified load module. (See Appendix C for more details.)
- If the DCB operand is written, the specified private library is searched for the load module. The link library is not searched.

EXAMPLES: In the following examples, EX1 gives control to a load module specified by the entry point DIVIDE. Because the DCB operand is not written, DIVIDE should be located in either the job library or the link library. The identifier, 41, is to be associated with this calling sequence.

EX2 gives control to a load module specified by the entry point COMPUTE. Because the DCB operand is not written, COMPUTE should be located in either the job library or the link library. When control is passed to COMPUTE, register 1 contains the address of a problem program parameter list. This list is part of the LINK macro-expansion, and it contains the two addresses TEMP1 and TEMP2+24. No identifier is to be associated with this calling sequence.

EX3 causes control to be passed to a load module that resides in the partitioned data set associated with the data control block located at PRILIB2. The load module is described by the list entry located at WORK1. A three-word problem program parameter list is generated in the macro-expansion; it contains the three addresses ARRAY3, I, and J. The location of the parameter list will be indicated by the contents of register 1 when the load module is given control. No identifier is to be associated with this calling sequence.

```
EX1   LINK      EP=DIVIDE,ID=41
EX2   LINK      EP=COMPUTE,PARAM=(TEMP1,TEMP2+24)
EX3   LINK      DE=WORK1,DCB=PRILIB2,PARAM=(ARRAY3,I,J)
```

L- AND E-FORM USE: The standard form of the LINK macro-instruction can result in a macro-expansion containing two parameter lists:

- A supervisor parameter list, which results from all operands of the macro-instruction except the PARAM and VL operands. This list is used by the supervisor to locate and acquire the specified load module.

- A problem program parameter list, which results from the optional PARAM operand. This list is identical to the parameter list resulting from a CALL macro-instruction, and is used to pass parameters to the specified load module.

The standard-form LINK macro-expansion can therefore consist of the following units of code:

Designation for Unit of Code and its Address	Code
AB	Branch to AE
ASPL	Supervisor parameter list
APL	Problem program parameter list
AE	Executable code terminated by an SVC instruction

When the LINK SVC instruction is executed, ASPL (the address of the supervisor parameter list) is in register 15, and APL (the address of the problem program parameter list) is in register 1.

Because both lists must be able to be remote, another special keyword operand, called the SF operand, is used in combination with the MF operand to provide nonstandard macro-instruction forms. The SF and MF operands can be written in the LINK macro-instruction as shown in the following format:

$$\left\{\begin{array}{l} SF=L \\ SF=(E,\left\{\begin{array}{l}spl\text{-}addrx\\(15)\end{array}\right\}) \\ MF=(E,\left\{\begin{array}{l}pl\text{-}addrx\\(1)\end{array}\right\})[,SF=(E,\left\{\begin{array}{l}spl\text{-}addrx\\(15)\end{array}\right\})] \end{array}\right\}$$

In the above format, spl specifies the address of a remote supervisor parameter list, and pl specifies the address of a remote problem program parameter list. If (15) or (1) is written as shown, the address of the remote list must be loaded into the designated register before execution of the macro-instruction.

Four SF and MF combinations are shown in the above format. These result in macro-expansions consisting of the following units of code:

SF and MF Combination	Units of Code in Macro-Expansion
SF=L	ASPL
SF=(E,ASPL)	AB,APL,AE
MF=(E,APL)	AB,ASPL,AE
MF=(E,APL),SF=(E,ASPL)	AE

The effect of each SF and MF combination is as follows:

- SF=L results in only a supervisor parameter list. Neither the PARAM nor the VL operand can be written in the macro-instruction.
- SF=(E,ASPL) results in a macro-expansion that does not contain a supervisor parameter list. Parameters in the remote supervisor parameter list can be dynamically changed as in a normal E-form macro-instruction.
- MF=(E,APL) specifies a normal E-form macro-instruction. Note that MF=L cannot be written in the LINK macro-instruction, but a remote problem program parameter list can be formed by using the L form of the CALL macro-instruction.
- MF=(E,APL),SF=(E,ASPL) indicates that both parameter lists are remote.

The LINK macro-instruction has one special operand requirement: the ID operand can be written in all forms except that specified by SF=L.

XCTL -- Transfer Control to a Load Module (S)

The XCTL macro-instruction gives control from the load module in which it appears to the load module it specifies. If the load module named by the entry point is reenterable and a copy is in main storage, it is used. If the load module is serially reusable, and if a copy is in main storage and not being used (in a type II linkage), it is used; if this copy is being used, the request for its use is queued. If the load module is not reusable and an unused copy is in main storage, it is used; if this copy has been used (in a type-II linkage), a new copy is loaded. If no copy of the load module is in main storage, a copy is loaded.

The module given control executes at the same level of control as the module issuing the XCTL macro-instruction. The main storage area occupied by the module issuing the XCTL macro-instruction may be freed for other uses. Therefore, the load module given control cannot return

control to the module that issued the XCTL macro-instruction. If the load module executes a RETURN macro-instruction, control is given to the next higher level load module, if the load module issuing the XCTL was not the highest level program of a task.

Before issuing an XCTL macro-instruction, a load module must restore the return register, the program mask, the program interruption control area, and registers 13 and 14 as they were upon entry to the load module. Registers 2 to 12 must be restored before the control program receives control. Either the user must provide the coding to restore the registers, or he can request that the expansion of the XCTL macro-instruction restore registers in the range 2 through 12 from the save area (originally pointed to by register 13 when the load module was given control). The save area is associated with the load module that is one level of control above that of the module issuing the XCTL.

Name	Operation	Operand
[symbol]	XCTL	[(reg_1-integer[,reg_2-integer])] ,{EP=symbol / EPLOC=addr / DE=addr}[,DCB=addr]

reg_1,reg_2
specifies the range of registers, from 2 through 12, that are to be restored. Reg_1 must be specified less than or equal to reg_2.

If this operand is omitted, the user is responsible for restoring the registers properly.

EP
specifies the symbolic name of an entry point in the load module to be given control.

The entry point name must either be a name contained in the directory of a partitioned data set (member name or alias) or have been identified to the control program through an IDENTIFY macro-instruction.

EPLOC
specifies the address of a double-word that contains the symbolic name of an entry point in the load module to be given control. The name must be left-justified in the double-word, and, if the name is less than eight characters, the double-word must be filled out with trailing blanks. The double-word can be aligned on a byte boundary and can be in the load module issuing the XCTL.

DE
specifies the address of the name field of a list entry describing the load module to be given control. The entry contains information previously extracted from the directory of a partitioned data set by a BLDL macro-instruction. (Refer to "Basic Partitioned Access Method" in Section 3 for a description of the BLDL macro-instruction.)

If the DE operand is written, the DCB operand must be identical to the DCB operand specified in the corresponding BLDL macro-instruction.

The list entry can be in the load module issuing the XCTL.

DCB

specifies the address of a data control block opened for a private library (partitioned data set) that is to be searched for the load module to which control is to be given. If the EP or EPLOC operand was written and the load module is not found in the library specified by the DCB operand, the link library is searched.

If the DCB operand is omitted, the load module is assumed to be in either the job library or the link library. The job library, if one exists, is searched first.

The data control block addressed by this operand must specify use of the EXCP macro-instruction, and must have been opened for INPUT before execution of the XCTL macro-instruction. (Refer to the publication IBM System/360 Operating System: System Programmer's Guide, Form C28-6550 for a description of the EXCP macro-instruction.) This data control block must not be used for any purpose other than the LINK, XCTL, LOAD, ATTACH, and BLDL macro-instructions. The data control block must not be in the load module issuing the XCTL, because the module can be overlaid during execution of the macro-instruction.

The data control blocks for the job and link libraries are always open.

CAUTIONS: During execution of the XCTL macro-instruction, the supervisor will abnormally terminate the task if the specified load module cannot be located.

If the XCTL macro-instruction is issued by a load module that was given control by a direct (type I) linkage from another load module, the main storage area occupied by the load module that made the direct linkage may be freed for other uses; the area occupied by the module issuing the XCTL will not be freed. This is because the module that made the direct linkage still appears to the supervisor to have control.

If the "only loadable" (OL) attribute was specified when the load module was processed by the linkage editor, an attempt to transfer control to the module will cause abnormal termination.

ENVIRONMENT: The following apply if option 4 was excluded from the system:

- An entry point identified to the control program in an IDENTIFY macro-instruction cannot be specified in an XCTL macro-instruction.
- The supervisor will abnormally terminate the task if the XCTL macro-instruction is issued by an asynchronous exit routine.
- If the DCB operand is written, the specified private library is searched for the load module. The link library is not searched.
- A XCTL macro-instruction specifying a load module that was not previously brought into main storage by a LOAD macro-instruction usually will load the specified load module. (See Appendix C for more details.)

EXAMPLES: In the following examples, EX1 passes control to a load module identified by the entry point MULT. Because the DCB operand is not written, MULT should be located in either the job library or link library. Registers 2 through 12 will be restored.

EX2 causes control to be passed to a load module that resides in the partitioned data set associated with the data control block at INPUT.

The load module to be given control is described by the list entry located at WORK1. Registers 2 through 9 will be restored.

```
EX1  XCTL      (2,12),EP=MULT
EX2  XCTL      (2,9),DE=WORK1,DCB=INPUT
```

PROGRAMMING NOTES: Refer to Appendix C for additional information on dynamic program management.

L- AND E-FORM USE: The standard form of the XCTL macro-instruction can result in a macro-expansion containing only a supervisor parameter list. This list results from all operands of the standard macro-instruction and is used to locate and acquire the specified load module.

The standard-form XCTL macro-expansion can therefore consist of the following units of code:

Designation for Unit of Code and Its Address	Code
AB	Branch to AE
ASPL	Supervisor parameter list
AE	Executable code terminated by an SVC instruction

When the XCTL SVC instruction is executed, ASPL (the address of the supervisor parameter list) is in register 15.

The nonstandard forms of the XCTL macro-instruction provide the only way of passing a problem program parameter list (designated APL) to the load module to be given control, because these forms allow the list to be remote. This list is identical to the parameter list resulting from a CALL macro-instruction. When the XCTL SVC instruction is executed, APL (address of the problem program parameter list) is in register 1.

Because both lists must be able to be remote, another special keyword operand, called the SF operand, is used in combination with the MF operand to provide nonstandard macro-instruction forms. The SF and MF operands can be written in the XCTL macro-instruction as shown in the following format:

$$\left\{\begin{array}{l} \text{SF=L} \\ \text{SF=(E,}\left\{\begin{array}{l}\text{spl-addrx}\\ \text{(15)}\end{array}\right\}\text{)} \\ \text{MF=(E,}\left\{\begin{array}{l}\text{pl-addrx}\\ \text{(1)}\end{array}\right\}\text{)[,SF=(E,}\left\{\begin{array}{l}\text{spl-addrx}\\ \text{(15)}\end{array}\right\}\text{)]} \end{array}\right\}$$

In the above format, spl specifies the address of a remote supervisor parameter list, and pl specifies the address of a remote problem program parameter list. If (15) or (1) is written as shown, the address of the remote list must be loaded into the designated register before execution of the macro-instruction.

If the MF operand is omitted, register 1 is not set to zero.

Four SF and MF combinations are shown in the above format. These result in macro-expansions consisting of the following units of code:

SF and MF Combination	Units of Code in Macro-Expansion
SF=L	ASPL
SF=(E,ASPL)	AE
MF=(E,APL)	AB,ASPL,AE
MF=(E,APL),SF=(E,ASPL)	AE

The effect of each SF and MF combination is as follows:

- SF=L results in only a supervisor parameter list. Only the operands of the standard-form XCTL macro-instruction can be written in this macro-instruction.

- SF=(E,ASPL) results in a macro-expansion that does not contain a supervisor parameter list. Parameters in the remote supervisor parameter list can be dynamically changed as in a normal E-form macro-instruction.

- MF=(E,APL) specifies a normal E-form macro-instruction. Note that MF=L cannot be written in the XCTL macro-instruction, but a remote problem program parameter list can be formed by using the L form of the CALL macro-instruction.

- MF=(E,APL),SF=(E,ASPL) indicates that both parameter lists are remote.

Thus, to change and add to the remote problem program parameter list, the MF=E form of the XCTL macro-instruction should be used. All operands are allowed in this form, including the PARAM and VL operands described in the LINK macro-instruction description. The ID operand, which is described in the LINK macro-instruction description, is not allowed.

LOAD -- Load and Retain a Load Module (R)

The LOAD macro-instruction acquires a specified load module and causes the supervisor to retain the module for use by the task issuing the LOAD. If a copy of the load module is not currently available in main storage, one is fetched. The module is associated with the requesting task and cannot be released until the task either terminates or uses a DELETE macro-instruction to release the module. However, the module can be used in a type-II linkage by any task of the job step whose task issued the LOAD macro-instruction. If the module is reenterable and from the link library, it can be used by any task of any job step. (Refer to Appendix C for a more complete discussion of the action of the LOAD macro-instruction.) Note that this macro-instruction does not initiate execution of the load module.

Name	Operation	Operand
[symbol]	LOAD	{EP=symbol / EPLOC={addrx / (0)} / DE={addrx / (0)}} [,DCB={addrx / (1)}]

EP
specifies the symbolic name of an entry point in the load module.

The entry point name must either be a name contained in the directory of a partitioned data set (member name or alias) or have been identified to the control program through the use of the IDENTIFY macro-instruction.

EPLOC
specifies the address of a double-word that contains the symbolic name of an entry point in the load module. The name must be

left-justified in the double-word, and, if the name is less than eight characters, the double-word must be filled out with trailing blanks. The double-word can be aligned on a byte boundary.

If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

DE

specifies the address of the name field of a list entry describing the load module to be acquired. The entry contains information previously extracted from the directory of a partitioned data set by a BLDL macro-instruction. (Refer to "Basic Partitioned Access Method" in Section 3 for the BLDL macro-instruction.)

If the DE operand is written, the DCB operand must be identical to the DCB operand specified in the corresponding BLDL macro-instruction.

If (0) is written, the list address must have been loaded into parameter register 0 before execution of this macro-instruction.

DCB

specifies the address of a data control block opened for a private library (partitioned data set) that is to be searched for the required load module. If the EP or EPLOC operand was written and the load module is not found in the library specified by the DCB operand, the link library is searched.

If the DCB operand is omitted, the load module is assumed to be in either the job library or the link library. The job library, if one exists, is searched first.

The data control block addressed by this operand must specify use of the EXCP macro-instruction, and must have been opened for INPUT before execution of the LOAD macro-instruction. (Refer to the publication IBM System/360 Operating System: System Programmer's Guide, Form C28-6550, for a description of the EXCP macro-instruction.) This data control block must not be used for any purpose other than the LINK, XCTL, LOAD, ATTACH, and BLDL macro-instructions.

The data control blocks for the job and link libraries are always open.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

The 24-bit actual address of the entry point is returned by the supervisor in register 0 after execution of the macro-instruction.

CAUTIONS: During execution of the LOAD macro-instruction, the supervisor will abnormally terminate the task issuing the LOAD if the specified load module cannot be located in the indicated source.

ENVIRONMENT: If option 4 was excluded from the system, an entry point identified to the control program in an IDENTIFY macro-instruction cannot be specified in a LOAD macro-instruction. A module loaded by an attached module is not automatically deleted when the latter module returns to a higher control level.

If option 4 was excluded and the DCB operand is written, the specified private library is searched for the load module. The link library is not searched.

EXAMPLES: In the following examples, EX1 causes the load module specified by entry point ALPHA to be loaded or located in storage, and the 24-bit address corresponding to ALPHA to be returned in register 0. If the load module is not in main storage, it will first be searched for in the job library for the job containing the LOAD and then, if not found, in the link library.

EX2 requests that the specified load module be made available. This module is described by the partitioned-data-set list entry located at DIRENT and is obtained from the data set associated with the data control block at PRILIB.

```
EX1  LOAD      EP=ALPHA
EX2  LOAD      DE=DIRENT,DCB=PRILIB
```

PROGRAMMING NOTES: Refer to Appendix C for additional information on the operation of the LOAD macro-instruction.

When a CALL macro-instruction is to execute a load module brought in by a LOAD, the entry operand of the CALL should be written as (15). Before execution of the CALL, the entry point address returned in register 0 during execution of the LOAD macro-instruction should be loaded into register 15. If a symbolic entry point name is written instead of (15), the specified program will be automatically linkage edited with the load module containing the CALL macro-instruction.

DELETE -- Delete a Retained Load Module (R)

The DELETE macro-instruction is used by a task to indicate to the supervisor that an in-storage copy of a load module is no longer required. This load module was previously acquired by the task by issuing a LOAD macro-instruction. The storage areas occupied by the load module are freed for other uses.

Name	Operation	Operand
[symbol]	DELETE	EP=symbol EPLOC={addrx | (0)} DE={addrx | (0)}

EP
: specifies the symbolic name of an entry point in the load module to be deleted.

EPLOC
: specifies the address of a double-word that contains the symbolic name of an entry point in the load module to be deleted. The name must be left-justified in the double-word, and, if the name is less than eight characters, the double-word must be filled out with trailing blanks. The double-word can be aligned on a byte boundary.

  If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

DE
: specifies the address of the name field of a list entry describing the load module to be deleted. The entry contains information

previously extracted from a directory of a partitioned data set by a BLDL macro-instruction. (Refer to "Basic Partitioned Access Method" in Section 3 for a description of the BLDL macro-instruction.)

If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

CAUTION: The name provided must be the same as the name given in a preceding LOAD macro-instruction (by an EP or EPLOC operand, or, indirectly, by a DE operand).

EXCEPTIONAL RETURNS: After execution of this macro-instruction, bits 24 through 31 of register 15 (the return code register) indicate the status of the operation. The hexadecimal code is as follows:

00 - successful completion
04 - the specified load module was not found

EXAMPLES: In the following examples, EX1 indicates that the load module specified by an entry point named ADD is no longer needed by the task issuing the DELETE. EX2 indicates that the load module described by the partitioned-data-set list entry located at STRING is no longer needed in storage.

```
EX1  DELETE    EP=ADD
EX2  DELETE    DE=STRING
```

IDENTIFY -- Identify an Embedded Entry Point (R)

The IDENTIFY macro-instruction is used by a task to inform the supervisor of an embedded entry point within a load module. This load module is one of the following:

- The load module that was last entered by the same task by means of a supervisor-assisted linkage. This includes the first load module of the job step, if it is still in control.
- A load module that was loaded by a LOAD macro-instruction issued by the same task.

After an IDENTIFY macro-instruction has been executed, the embedded entry point can be referred to by an ATTACH, LINK, XCTL, or LOAD macro-instruction. The subprogram having the specified entry point is assumed to be reenterable. The IDENTIFY macro-instruction is needed to specify an entry point only if the entry was not specified to the linkage editor as a member name or alias.

Name	Operation	Operand
[symbol]	IDENTIFY	{EP=symbol / EPLOC={addrx / (0)}},ENTRY={addrx / (1)}

EP
specifies the symbolic name of the entry point being identified to the supervisor.

EPLOC
specifies the address of a double-word that contains the symbolic name of the entry point being identified to the supervisor. The

name must be left-justified in the double-word, and, if the name is less than eight characters, the double-word must be filled out with trailing blanks. The double-word can be aligned on a byte boundary.

If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

ENTRY
specifies the address of the entry point being identified to the supervisor.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTION: Execution of this macro-instruction is not successful if it is issued by an asynchronous exit routine.

EXCEPTIONAL RETURNS: After execution of this macro-instruction, bits 24 through 31 of register 15 (the return code register) indicate the status of the operation. The hexadecimal code is as follows:

- 00 - successful completion
- 04 - an IDENTIFY that specified the same entry point name and address was issued previously
- 08 - the entry point name is the same as the name of a load module currently in main storage
- 0C - the entry point is not in the load modules that were searched
- 10 - the IDENTIFY was issued by an asynchronous exit routine.
- 14 - an IDENTIFY that specified the same entry point name but a different address was issued previously.

Codes 08, 0C, 10, and 14 are considered error conditions.

ENVIRONMENT: The full services of the IDENTIFY macro-instruction are available if option 4 was included in the system. If option 4 was excluded, option 3 permits the identification of an embedded entry point subject to the following rules:

- The entry point can be referred to only by an ATTACH macro-instruction.
- The identified subprogram cannot in turn issue another IDENTIFY macro-instruction.
- The entry point can be either in the module issuing the IDENTIFY macro-instruction or in a module loaded (using a LOAD macro-instruction) by any module given control in the job step.

If both option 3 and 4 were excluded, the IDENTIFY macro-instruction will be treated as a NOP at execution.

EXAMPLE: In the following example, EX1 informs the supervisor of an additional entry point within a load module that either was last entered by the task as a result of a supervisor-assisted linkage or was loaded by the LOAD macro-instruction. The entry point's name is COSINE and its address is in register 6.

```
EX1  IDENTIFY    EP=COSINE,ENTRY=(6)
```

PROGRAMMING NOTES: A load module can be retrieved by its member name or its aliases defined at linkage edit time. If additional entry points need to be defined at execution time, the IDENTIFY macro-instruction should be used.

Refer to Appendix C for additional information on dynamic program management.

GETMAIN -- Allocate Main Storage (R)

The R form of the GETMAIN macro-instruction requests that the supervisor dynamically allocate a single area of main storage for task use. The FREEMAIN macro-instruction can be used to release the allocated storage for other uses; otherwise, task termination automatically releases all of the allocated storage owned by the task.

The R form of the GETMAIN macro-instruction can be used upon entry to a reenterable program to obtain main storage for a save area or for other uses. When the macro-instruction is executed, the supervisor will not modify the contents of the save area pointed to by register 13.

Name	Operation	Operand
[symbol]	GETMAIN	R,{LV=value[,SP=value] / LV=(0)}

R
: specifies that this is the R form of the macro-instruction, and that a single area of main storage is requested.

LV
: specifies the length, in bytes, of the requested area of main storage. The value should be a multiple of eight; if it is not, the supervisor will act as though the next higher multiple of eight had been written. The length of the area cannot exceed the maximum established at system generation time.

  If (0) is written, the length must have been loaded into the three low-order bytes of parameter register 0, and the subpool number into the high-order byte, before execution of this macro-instruction.

SP
: specifies the number of the subpool from which the requested storage area is to be allocated. If the specified subpool does not exist, a new subpool is created. If the specified subpool does exist but does not have enough unallocated space for the area specified, it is extended. If this operand is omitted and LV=(0) is not written, subpool zero is assumed. The value of this operand must be from 0 to 127.

The address of the allocated storage area is returned by the supervisor in the three low-order bytes of register 1. The high-order byte is zero.

When this macro-instruction is executed, the supervisor allocates storage for the requested area. The area begins on a double-word boundary and is assigned from a storage block having the task's storage protection key.

CAUTIONS: The storage area is not cleared to zero when allocated.

During execution of the GETMAIN macro-instruction, the supervisor will abnormally terminate the task issuing the GETMAIN if one of the following occurs:

- The subpool number exceeds 127.
- More storage is requested than can be allocated as a single area, or can be made available by the system.

ENVIRONMENT: If option 4 was excluded from the system, there is one unnumbered subpool; the SP operand is therefore ignored. Any request for more main storage than is immediately available causes abnormal termination of the job step if (1) option 14 was excluded from the system, or (2) there is no lower priority job step being executed concurrently.

If option 14 was excluded from the system, but option 4 was included, a request for more main storage than is immediately available causes the requesting task to wait until sufficient storage is released by other tasks.

If option 4 was excluded, other options specified at system generation time determine conditions that may cause abnormal termination of a job step. For descriptions of these conditions, refer to the publication IBM System/360 Operating System: Messages and Completion Codes, Form C28-6608.

EXAMPLES: In the following examples, EX1 requests 100 bytes of storage from subpool zero. Note that actually 104 bytes will be allocated, because of the multiple-of-eight rule. EX2 requests 110 bytes from subpool 10; in this case, 112 bytes will be allocated. EX3 indicates that register zero has been loaded with the subpool number and the length of the requested area.

```
EX1  GETMAIN   R,LV=100
EX2  GETMAIN   R,LV=100+2*5,SP=10
EX3  GETMAIN   R,LV=(0)
```

PROGRAMMING NOTES: The supervisor allocates the area from the specified subpool. Thus, through the SP operand, a task organizes allocated storage into subpools. Each task can have as many as 128 subpools, each identified by an integer from 0 to 127, called a "subpool number." A new subpool is automatically created when a task makes a request for storage and specifies a new subpool number from 1 to 127 by means of the SP operand. Subpool 0 is automatically created by the control program when the first task of the job step is created.

When a subpool is created, storage is assigned to it in multiples of 2048-byte blocks. The number of blocks assigned is the minimum number needed to satisfy the storage request. These blocks are contiguous in storage. Once a subpool has been created in this way, further storage requests specifying the same subpool number either are satisfied from the originally assigned storage blocks, or extend the size of the particular subpool by causing additional 2048-byte blocks to be assigned. The additional storage blocks may or may not be contiguous in storage with the original blocks.

A request for more storage than is immediately available is optionally filled by allocating storage areas assigned to lower priority job steps. The contents of these areas are saved in external storage and restored in last-out/first-in order. If no lower priority job step exists, however, the requesting task is placed in a wait condition until sufficient storage is released by other job steps, or by other tasks of the same job step.

Through the ATTACH macro-instruction, a task can give subpools to a subtask or can share subpools with one or more of its subtasks.

All subpools of two tasks, except subpool 0, are distinct from each other, even when they have the same identifying numbers, unless an ATTACH macro-instruction specifies sharing of the subpools. Subpool 0 is automatically shared by all tasks of the job step.

Storage owned or shared by a task can be released for other uses by a FREEMAIN macro-instruction. A subpool is automatically released when its owning task terminates.

GETMAIN -- Allocate Main Storage (S)

The S form of the GETMAIN macro-instruction requests that the supervisor dynamically allocate one or more areas of main storage for task use. The FREEMAIN macro-instruction can be used to release the allocated storage for other uses; otherwise, task termination automatically releases all of the allocated storage that is owned by the task.

Name	Operation	Operand
[symbol]	GETMAIN	{mode-{EU|EC},LV=value / mode-{VU|VC|LU|LC},LA=addr},A=addr[,SP=value]

mode
: specifies that this is the S form of the macro-instruction and specifies the type of request for storage. The meanings of the characters in this operand are:

  - E (element) specifies a request for a single area of main storage of a specific length.
  - V (variable) specifies a request for a single area of main storage whose length is between two values.
  - L (list) specifies a request for one or more areas of main storage. Each area is of a specific length.
  - U (unconditional) specifies that the request must be satisfied before the task can continue.
  - C (conditional) specifies that the request is not essential to the continuation of the task. If the entire request is satisfied, register 15 (the return code register) contains zero; if the request cannot be satisfied immediately, it is canceled and register 15 contains 4.

  Note that only the combinations of characters shown in the format description are allowed.

LV
: specifies the length, in bytes, of the single area of main storage requested by an EU or EC operand. The value should be a multiple of eight; if it is not, the supervisor will act as though the next higher multiple of eight had been written.

LA
: specifies the address of a user-provided list of lengths, in bytes, for the main storage areas requested. Each list entry must be four bytes long and must begin on a full-word boundary. The list must consist of:

  - For VU or VC requests: two 4-byte entries. The first entry specifies the minimum length needed by the task; the second entry specifies the maximum length that can be used. In each entry, the three low-order bytes contain the length, and the high-order byte is zero.

- For LU or LC requests: one 4-byte entry for each main storage area requested. Each entry specifies the length of a desired area. In each entry, the three low-order bytes contain the length. The high-order byte is zero, except in the last entry where the sign bit must be set to one.

All lengths specified should be in multiples of eight; if any length is not, the supervisor will act as though the next higher multiple of eight had been written.

A

specifies the address of a user-provided list in which the supervisor stores specifications of the main storage area or areas allocated for this request. Each list entry must be four bytes long and begin on a full-word boundary. The list must consist of:

- For EU or EC requests: one 4-byte entry in which the supervisor stores the address of the single main storage area allocated.

- For VU or VC requests: two 4-byte entries. In the first entry, the supervisor stores the address of the single main storage area allocated. In the second entry, the supervisor stores the length of the area.

- For LU or LC requests: one 4-byte entry for each main storage area requested by the LA operand. The supervisor stores in each entry the address of the main storage area allocated for the corresponding LA list entry.

The resulting parameter list can serve as valid input to a FREEMAIN macro-instruction.

SP

specifies the number of the subpool from which the requested storage area is to be allocated. If the specified subpool does not exist, a new subpool is created. If the specified subpool does exist but does not have enough unallocated space for the areas specified, it is extended. If this operand is omitted, subpool zero is used. The value of this operand must be from 0 to 127.

Note that all areas that satisfy an LU or LC request are allocated from the same subpool.

When this macro-instruction is executed, the supervisor allocates storage for the requested area or areas. Each area begins on a double-word boundary and is allocated from a block having the task's storage protection key.

CAUTIONS: Storage areas are not cleared to zero when allocated.

Before this macro-instruction is executed, the user must create the list for the LA operand, if used, and must provide the space for the A operand list. The lists for the LA and A operands must not have any common locations.

During execution of the GETMAIN macro-instruction, the supervisor will abnormally terminate the task issuing the GETMAIN if one of the following occurs:

- The subpool number exceeds 127.

- More storage is requested, unconditionally, than can be allocated as a single area, or can be made available by the system.

- The lists for the LA and A operands have common locations.

An error may occur in the amount of storage that is allocated if the maximum length specified in a VU or VC request exceeds $2^{24}-8$. After being rounded to a multiple of 8, the requested maximum is truncated to include only the 24 low order bits. A request for $2^{24}-7$ bytes, for example, would be rounded upward to 2^{24} and then truncated to a value of 0.

EXCEPTIONAL RETURNS: After execution of the EC, VC, and LC types of this macro-instruction, bits 24 through 31 of register 15 (the return code register) indicate the status of the operation. The hexadecimal code is as follows:

00 - request was satisfied
04 - request was not satisfied

ENVIRONMENT: If option 4 was excluded from the system, the GETMAIN macro-instruction can be used to request only a single area of main storage. The job step is abnormally terminated if the mode operand is written as LU or LC. There is one unnumbered subpool; the SP operand is therefore ignored. Any unconditional request for more main storage than is immediately available causes abnormal termination of the job step if (1) option 14 was excluded from the system, or (2) there is no lower priority job step being executed concurrently.

If option 14 was excluded from the system but option 4 was included, a request for more storage than is immediately available causes the requesting task to wait until sufficient storage is released by other tasks.

If option 4 was excluded, other options specified at system generation time determine conditions that may cause abnormal termination of a job step. For descriptions of these conditions, refer to the publication IBM System/360 Operating System: Messages and Completion Codes, Form C28-6608.

EXAMPLES: In the following examples, EX1 requests a 100-byte area in subpool 3. Note that actually 104 bytes will be allocated, because of the multiple-of-eight rule. The supervisor is to store the address of the area in the full-word at ALPHA. The area must be allocated before the task can continue processing.

EX2 requests one storage area in subpool 0. The minimum length needed is in the full-word at BETA and the maximum length that can be used is in the full-word at BETA+4. The area must be allocated before the task can continue processing. The supervisor is to store the address of the area in the full-word at GAMMA, and the length of the area actually allocated in the full-word at GAMMA+4.

EX3 requests several storage areas in subpool 4. The lengths of the desired areas are in a list at DELTA. The areas need not be allocated for processing to continue. If the supervisor allocates all the storage requested, it is to store the addresses of the areas in a list beginning at EPSILON.

```
EX1  GETMAIN   EU,LV=100,A=ALPHA,SP=3
EX2  GETMAIN   VU,LA=BETA,A=GAMMA
EX3  GETMAIN   LC,LA=DELTA,A=EPSILON,SP=4
```

PROGRAMMING NOTES: The supervisor allocates the one or several areas from the specified subpool. Thus, through the SP operand, a task organizes allocated storage into subpools. Each task can have as many as 128 subpools, each identified by an integer from 0 to 127, called a "subpool number." A new subpool is automatically created when a task makes a request for storage and specifies a new subpool number from 1 to 127 by means of the SP operand. Subpool 0 is automatically created by the control program when the first task of the job step is created.

When a subpool is created, storage is assigned to it in multiples of 2048-byte blocks. The number of blocks assigned is the minimum number needed to satisfy the storage request. These blocks are contiguous in storage. Once a subpool has been created in this way, further storage requests specifying the same subpool number either are satisfied from the originally assigned storage blocks, or extend the size of the particular subpool by causing additional 2048-byte blocks to be assigned. The additional storage blocks may or may not be contiguous in storage with the original blocks.

For an LU or LC request that is creating a new subpool, the first entry in the list is treated as the original request and subsequent entries are treated as further requests. Therefore, the storage blocks assigned to the subpool to satisfy a list request may or may not be contiguous.

An unconditional request for more storage than is immediately available is optionally filled by allocating storage areas assigned to lower priority job steps. The contents of these areas are saved in external storage and restored in last-out/first-in order. If no lower priority job step exists, however, the requesting task is placed in a wait condition until sufficient storage is released by other job steps, or by other tasks of the same job step.

After a conditional request, the contents of register 15 (the return code register) indicate whether or not the storage was allocated. Register 15 contains zero if the total request was satisfied, or four if it was not. An EC request is satisfied when the single block requested is allocated; a VC request, when at least the minimum storage requested is allocated; and an LC request, when every area requested is allocated. Note that, for an LC request, only if all areas can be allocated is any storage allocated.

The contents of register 15 are undefined after unconditional requests.

The supervisor places the address of each allocated storage area in the three low-order bytes of a full word in the list specified by the A operand. The high-order byte of each full word is reserved. For a VU or VC request, the length of each allocated block is placed in the list.

Through the ATTACH macro-instruction, a task can give subpools to a subtask or can share subpools with one or more of its subtasks.

All subpools of two tasks, except subpool 0, are distinct from each other, even when they have the same identifying numbers, unless an ATTACH macro-instruction specifies sharing of the subpools. Subpool 0 is automatically shared by all tasks of the job step.

Storage owned or shared by a task can be released for other uses by a FREEMAIN macro-instruction, which can use, without modification, the lists specified in the A and LA operands of the GETMAIN macro-instruction. A subpool is automatically released when its owning task terminates.

L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B. The L form can be used to form a remote parameter list for use by E forms of both the GETMAIN and FREEMAIN macro-instructions.

FREEMAIN -- Release Allocated Main Storage (R)

The R form of the FREEMAIN macro-instruction releases one subpool or one main storage area previously acquired through one or more GETMAIN macro-instructions. The save area pointed to by register 13 is not modified.

In the following format description, the first format is for release of one main storage area; the second format is for release of one subpool.

Name	Operation	Operand
[symbol]	FREEMAIN	R,{LV=value[,SP=value] / LV=(0)},A={addrx / (1)}
[symbol]	FREEMAIN	R,SP={value / (0)}

R
specifies that this is the R form of the macro-instruction.

LV
specifies the length, in bytes, of the main storage area to be released. The value should be a multiple of eight; if it is not, the supervisor will act as though the next higher multiple of eight had been written.

If (0) is written, the length must have been loaded into the three low-order bytes of parameter register 0 and the subpool number into the high-order byte, before execution of this macro-instruction.

SP (for release of a main storage area)
specifies the number of the subpool from which the main storage area is to be released. If this operand is omitted and if LV=(0) is not written, subpool zero is assumed. The value of this operand must be from 0 to 127.

A
specifies the address of a full-word whose three low-order bytes contain the address of the storage area to be released. The address of the storage area must be a multiple of eight.

If (1) is written, the address of the storage area itself (not the address of a full-word containing the storage area address) must have been loaded into the three low-order bytes of parameter register 1 before execution of this macro-instruction.

SP (for release of a subpool)
specifies the number of the subpool to be released. Subpool zero cannot be released.

If (0) is written, the subpool number must have been loaded into the high-order byte of parameter register 0 before execution of this macro-instruction. The three low-order bytes must be zeros.

CAUTIONS: During execution of the FREEMAIN macro-instruction, the supervisor will abnormally terminate the task issuing the FREEMAIN if one of the following occurs:

- The subpool number exceeds 127.
- The storage area to be released is not in the specified subpool.
- The address of an area to be released is not a multiple of eight.
- The storage area to be released was not allocated to the task, or was released previously.

ENVIRONMENT: If option 4 was excluded from the system, there is one unnumbered subpool; the SP operand is therefore ignored.

If option 4 was excluded, other options specified at system generation time determine conditions that may cause abnormal termination of a job step. For descriptions of these conditions, refer to the publication IBM System/360 Operating System: Messages and Completion Codes.

EXAMPLES: In the following examples, EX1 requests the release from subpool zero of a 16-byte area whose address is in register 1. EX2 requests the release from subpool 5 of a 322-byte area, whose address is in the full-word at ADD. Note that actually 328 bytes will be released, because of the multiple-of-eight rule. EX3 requests release of the entire subpool 99.

```
EX1  FREEMAIN  R,LV=16,A=(1)
EX2  FREEMAIN  R,LV=322,SP=5,A=ADD
EX3  FREEMAIN  R,SP=99
```

FREEMAIN -- Release Allocated Main Storage (S)

The S form of the FREEMAIN macro-instruction releases one or more areas of main storage previously acquired through one or more GETMAIN macro-instructions.

Name	Operation	Operand
[symbol]	FREEMAIN	{ mode-E,LV=value / mode-V / mode-L,LA=addr } ,A=addr[,SP=value]

mode
: specifies that this is the S form of the macro-instruction and specifies the type of storage release being requested. The mode operand is written as one of the following characters:

  E (element) specifies release of a single storage area whose length is given in the LV operand.
  V (variable) specifies release of a single storage area whose length is given by means of the A operand.
  L (list) specifies release of one or more storage areas whose lengths are given by means of the LA operand.

LV
: specifies the length, in bytes, of the single area of main storage to be released. The value should be a multiple of eight; if it is not, the supervisor will act as though the next higher multiple of eight had been written.

LA

specifies the address of a user-provided list of lengths, in bytes, of the main storage areas to be released. Each list entry is four bytes long, and must begin on a full-word boundary. Each full-word contains a length specification in its three low-order bytes and, except for the last entry, a zero in its high-order byte. The sign bit in the last entry must be 1. Each length should be a multiple of eight; if it is not, the supervisor will act as though the next higher multiple of eight had been written.

A

specifies the address of a user-provided list that specifies the one or more main storage areas to be released. Each list entry must be four bytes long, and must begin on a full-word boundary. The list must consist of:

- For E releases: one 4-byte entry containing the address of the single main storage area to be released. The address is in the three low-order bytes of the entry, and the high-order byte is zero.

- For V releases: two 4-byte entries. The first entry contains the address of the single main storage area to be released. The second contains the length of the area. In each entry, the address or length is in the three low-order bytes, and the high-order byte is zero.

- For L releases: one 4-byte entry for each main storage area whose length is given in the list referred to by the LA operand. In each entry, the address is in the three low-order bytes, and the high-order byte is zero. The first address in this list and the first length in the LA list together specify the first area, the second address and length specify the second area, etc.

All addresses must be multiples of eight.

SP

specifies the number of the subpool from which the main storage area or areas are to be released. If this operand is omitted, subpool zero is assumed. The operand's value must be from 0 to 127. All areas in an L release must be in the same subpool.

CAUTIONS: Before this macro-instruction is executed, the user must create the list for the LA operand, if used, and for the A operand. The lists for LA and A operands must not have any common locations.

During execution of the FREEMAIN macro-instruction, the supervisor will abnormally terminate the task issuing the FREEMAIN if one of the following occurs:

- The subpool number exceeds 127.
- The storage area to be released is not in the specified subpool.
- The address of an area to be released is not a multiple of eight.
- The storage area to be released was not allocated to the task, or was released previously.
- The lists for the LA and A operands have common locations.

ENVIRONMENT: If option 4 was excluded from the system, the FREEMAIN macro-instruction can be used to release only a single area of main storage. Since there is one unnumbered subpool, the SP operand is ignored. The job step is abnormally terminated if the mode operand is written as L.

If option 4 was excluded, other options specified at system generation time determine conditions that may cause abnormal termination of a job step. For descriptions of these conditions, refer to the publication IBM System/360 Operating System: Messages and Completion Codes.

EXAMPLES: In the following examples, EX1 requests the release of a 120-byte storage area from subpool zero. The address of the area is contained in the full-word at BLOKADD.

EX2 requests the release of a storage area from subpool 10. The address and length of the area are contained in the full-words at BLOKDESC and BLOCKDESC+4, respectively.

EX3 requests the release of several storage areas from subpool zero. The consecutive full-words beginning at location LENGTHS specify the number of bytes in each area. The consecutive full-words beginning at location LOCS specify the addresses of the areas.

```
EX1  FREEMAIN  E,LV=120,A=BLOKADD
EX2  FREEMAIN  V,A=BLOKDESC,SP=10
EX3  FREEMAIN  L,LA=LENGTHS,A=LOCS
```

PROGRAMMING NOTES: If a single storage area is to be released, some or all of the area specifications (address, length, and subpool number) can reside in the area to be released. If several areas are to be released, the list for the LA operand or the list for the A operand, or both, can reside in the areas to be released, provided that:

- Neither list begins in the first word of the first area to be released.
- A list within an area lies wholly within the area.

The lists specified in the A and LA operands can also be those specified in the corresponding operands of the GETMAIN macro-instruction that caused the storage to be allocated. No modification of these lists is necessary.

L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B. The E form can use the remote parameter list formed by the L form of either the FREEMAIN or the GETMAIN macro-instruction. No error results from the specification of mode as conditional or unconditional in the L-Form GETMAIN macro-instruction.

ATTACH -- Create and Attach a Task (S)

The ATTACH macro-instruction creates a new task and initiates execution of the new task by passing control to a specified entry point in a program. If the load module named by the entry point is reenterable and a copy is in main storage, it is used. If the load module is serially reusable, and if a copy is in main storage and not being used (in a type II linkage), it is used; if this copy is being used, the request for its use is queued. If the load module is not reusable and an unused copy is in main storage, it is used; if this copy has been used (in a type-II linkage) a new copy is loaded. If no copy of the load module is in main storage, a copy is loaded.

The task created is a subtask of the task whose program executed the ATTACH macro-instruction. It competes with the task issuing the ATTACH macro-instruction, as well as with other active tasks, for system resources. The subtask can terminate itself by means of either an ABEND macro-instruction or a RETURN macro-instruction executed in its highest level program.

The ATTACH macro-instruction allows the task to give main storage subpools to or share them with the subtask. It also allows the task to set the limit and dispatching priorities of the subtask. The macro-instruction specifies the way in which the task is to be notified of the subtask's termination.

Name	Operation	Operand
[symbol]	ATTACH	{EP=symbol / EPLOC=addr / DE=addr}[,DCB=addr][,PARAM=({addr,}...)]
		[,VL=1]][,ECB=addr][,{GSPV=value / GSPL=addr}]
		[,{SHSPV=value / SHSPL=addr}][,ETXR=addr]
		[,LPMOD=value][,DPMOD=value]

EP
: specifies the symbolic name of the entry point in the load module in which execution of the subtask is to begin.

  The entry point must either be a name contained in the directory of a partitioned data set (member name or alias) or have been identified to the control program through the use of the IDENTIFY macro-instruction.

EPLOC
: specifies the address of a double-word that contains the symbolic name of the entry point in the load module in which execution of the subtask is to begin. The name must be left-justified in the double-word, and, if the name is less than eight characters, the double-word must be filled out with trailing blanks. The double-word can be aligned on a byte boundary.

DE

specifies the address of the name field of a list entry describing the first load module to be executed. The entry contains information previously extracted from the directory of a partitioned data set by a BLDL macro-instruction. (Refer to "Basic Partitioned Access Method" in Section 3 for a description of the BLDL macro-instruction.)

If the DE operand is written, the DCB operand must be identical to the DCB operand specified in the corresponding BLDL macro-instruction.

DCB

specifies the address of a data control block opened for a private library (partitioned data set) that is to be searched for the specified load module. If the EP or EPLOC operand was written and the load module is not found in the private library specified by the DCB operand, the link library is searched.

If the DCB operand is omitted, the load module is assumed to be in either the job library or the link library. The job library, if one exists, is searched first.

The data control block addressed by this operand must specify use of the EXCP macro-instruction, and must have been opened for INPUT before execution of the ATTACH macro-instruction. (Refer to the publication IBM System/360 Operating System: System Programmer's Guide, Form C28-6550 for a description of the EXCP macro-instruction.) This data control block must not be used for any purpose other than the LINK, XCTL, LOAD, ATTACH, and BLDL macro-instructions.

The data control blocks for the job and link libraries are always open.

PARAM

specifies, as a sublist, address parameters to be passed from the task to the new subtask. If one or more operands are written in the sublist, a problem program parameter list is generated. It consists of a full-word for each operand. Each full-word is aligned on a full-word boundary and contains, in its three low-order bytes, the address to be passed. The addresses appear in the parameter list in the same order as in the macro-instruction.

When the specified load module is entered, register 1 (the parameter list register) contains the address of the problem program parameter list.

If the PARAM operand is omitted, register 1 is not set to zero.

VL

specifies that the sign bit is to be set to 1 in the last full-word in the problem program parameter list.

The parameter list has a fixed length if it is to contain a certain, known number of parameters every time the subtask is given control. The list has a variable length if it can contain a varying number of parameters. Only in the latter case should the VL operand be written in order to mark the end of the list.

If the list has a variable length and if register notation is used to write out the last PARAM address, the user's problem program can set the sign bit in the designated register to 1. If this is done, the VL operand need not be written.

ECB

specifies the address of an event control block (ECB) representing completion of the subtask. When the operand is written, upon completion of the task, the event control block is automatically posted by the supervisor. If the task completes normally (as indicated by a RETURN macro-instruction), the return code in register 15 is used as the post code. If the task completes abnormally (as indicated by an ABEND macro-instruction), the completion code specified in the ABEND macro-instruction is used as the postcode. (Refer to the WAIT and POST macro-instructions in "Task Synchronization" for a discussion of the event control block and its use.)

GSPV

specifies a single subpool number. The subpool, if owned by the task issuing the ATTACH macro-instruction, is given to the new subtask. If shared by the task, the subpool is shared between the new subtask and those other tasks with which the task issuing the ATTACH previously shared the subpool. The task issuing the ATTACH no longer shares the subpool and cannot in the future either give or share this subpool with another new subtask. (Refer to "Programming Notes" below for further explanation of the giving of subpools.)

GSPL

specifies the address of a list of subpool numbers. Each subpool in the list must be either owned or shared by the task issuing the ATTACH macro-instruction. Each subpool to be shared is treated as the single subpool specified by the GSPV operand, which is described above. The first byte of the list contains the number of the remaining bytes in the list, and the subsequent bytes contain subpool numbers.

SHSPV

specifies the number of a single subpool to be shared with the new subtask. This subpool must be owned or shared by the task issuing the ATTACH macro-instruction.

SHSPL

specifies the address of a list of subpool numbers. The subpools must be owned or shared by the task issuing the ATTACH macro-instruction. They are shared with the new subtask. The first byte of the list contains the number of the remaining bytes in the list, and the subsequent bytes contain subpool numbers.

ETXR

specifies the address of an exit routine in the task issuing the ATTACH macro-instruction. The routine is entered asynchronously, whether the subtask terminates normally or abnormally. (Refer to the ABEND macro-instruction in "Exceptional Condition Handling" for additional information on when the ETXR exit routine is entered.)

LPMOD

specifies an absolute value to be subtracted from the current limit priority of the task issuing the ATTACH macro-instruction. The number resulting from the subtraction is the limit priority for the new subtask. If the subtraction produces a negative number, the supervisor assigns a limit priority of zero. If this operand is omitted, the current limit priority of the attaching task is given to the new subtask.

DPMOD

specifies a signed value to be algebraically added to the current dispatching priority of the task issuing the ATTACH macro-instruction. The resulting number is the dispatching priority for

the new subtask. If the addition produces a negative number, the supervisor assigns a dispatching priority of zero to the subtask. If the addition produces a number greater than the subtask's limit priority, as computed from the LPMOD operand, the supervisor sets the dispatching priority equal to the limit priority. If this operand is omitted, the subtask's dispatching priority is set equal to either the limit priority of the subtask or the current dispatching priority of the task issuing the ATTACH, whichever is smaller. If this operand is written as an absolute expression, it can begin with a minus sign.

If register notation is used, a negative value is represented in the register in 32-bit two's complement form.

CAUTION: After issuing the ATTACH macro-instruction, the task must not modify the following information until the times indicated:

1. Data control block specified by the DCB operand: until execution of the subtask has begun.

2. Parameter list specified by the PARAM operand: until an arbitrary point has been reached in the subtask's execution.

The WAIT and POST macro-instructions and an event control block can be used as an interlock to signal when the data control block or problem program parameter list can be modified by the task.

If the task issuing the ATTACH macro-instruction attempts to terminate normally while any of its subtasks are still being processed, the task and all of its subtasks are terminated abnormally.

During execution of the ATTACH macro-instruction, the supervisor will abnormally terminate the new subtask if the load module with which execution of the subtask is to begin cannot be located.

If the "only loadable" (OL) attribute was specified when the load module was processed by the linkage editor, an attempt to attach the module will cause abnormal termination.

ENVIRONMENT: The following apply if option 4 was excluded from the system:

- The ATTACH macro-instruction results in the load module being executed serially, as though a LINK macro-instruction had been issued. No subtask or task control block is created. Upon normal termination of the attached load module, the event control block, if specified, is posted and the ETXR routine, if specified, is given control. When the attached load module and the ETXR routine (if specified) terminate normally, control is returned to the instruction that follows the ATTACH macro-instruction. Register 1 will contain zero upon entry to the ETXR routine and upon return to the attaching module. Abnormal termination of the attached load module causes termination of the job step.

- The following operands in the ATTACH macro-instruction are ignored:

  GSPV, GSPL, SHSPV, SHSPL } subpool operands

  LPMOD, DPMOD } priority operands

- If the ETXR routine is specified more than once, it is assumed to be reenterable. Because attached modules are executed serially, concurrent executions of an ETXR routine can occur only if an ATTACH macro-instruction is issued within one or more asynchronous exit routines.

- A RETURN macro-instruction issued by an attached module will not:

  a. Free allocated main storage obtained by the module.

  b. Delete any load module retained by a LOAD macro-instruction that was issued by the module.

  c. Close any data sets opened by the module.

- If the DCB operand is written in the ATTACH macro-instruction, only the specified private library is searched for the load module. The link library is not searched.

- An ATTACH macro-instruction specifying a load module that was not previously brought into main storage by a LOAD macro-instruction usually will load the specified load module. (See Appendix C for details.)

EXAMPLES: In the following examples, EX1 creates a subtask whose execution is to begin with the load module specified by the entry point ADD. The supervisor is to search for this load module in the job library defined for the job containing the task. If not found in the job library, the module is to be searched for in the installation's link library. No parameters are to be passed to the subtask. No subpools are to be given or shared with the subtask, other than subpool zero, which is always shared. The subtask is to have the same limit and dispatching priority values as the task. No event control block or exit routine is provided.

EX2 creates a subtask whose execution is to begin with the load module specified by the entry point MULT. The module is in the partitioned data set associated with the data control block located at DCB35. A problem program parameter list is generated as part of the macro-expansion. This list consists of three full-words, containing the three addresses T1, T2, and T3. When the subtask is given control, register 1 will contain the address of the parameter list. An event control block located at DONE is to be posted when the subtask completes. No subpools are to be given to or shared with the subtask, other than subpool zero. The subtask is to have the same limit and dispatching priority values as the task. No exit routine is provided.

EX3 creates a subtask whose execution is to begin with the load module described in an in-storage list entry located at DSTRING. This description was previously obtained by executing a BLDL macro-instruction that referred to the data control block located at DCB36. The task is giving subpool 1 to the subtask; other subpools, specified in the list beginning at location SPLIST, are to be shared by the task and subtask. The subtask's limit priority is to be the same as that of the task's. The subtask's dispatching priority is to be three units less than that of the task. No parameters are to be passed to the subtask. No event control block or exit routine is provided.

EX4 creates a subtask whose execution is to begin with the load module specified by the entry-point name in the double-word located at NAME. The supervisor is to search for this load module in the job library defined for the job containing the task. If not found in the job library, the module is to be sought in the installation's link library. An in-storage problem program parameter list is generated as part of the macro-expansion. This list consists of a single full-word

containing the address ARRAY1. No subpools are to be given or shared with the subtask, other than subpool zero. The subtask is to have the same limit and dispatching priority values as the task. An event control block is not provided. When the subtask terminates, control is to be passed asynchronously to the location TERMINAL.

```
EX1     ATTACH    EP=ADD
EX2     ATTACH    EP=MULT,DCB=DCB35,PARAM=(T1,T2,T3),ECB=DONE
EX3     ATTACH    DE=DSTRING,DCB=DCB36,GSPV=1,SHSPL=SPLIST,DPMOD=-3
EX4     ATTACH    EPLOC=NAME,PARAM=(ARRAY1),ETXR=TERMINAL
```

PROGRAMMING NOTES: When this macro-instruction is executed, the supervisor creates a subtask by creating a task control block (TCB) from the information given in the ATTACH macro-instruction. The address of the subtask's task control block is returned to the task in register 1 after execution of the ATTACH. The task can save the task control block address for use in operands of other system macro-instructions, such as DETACH and EXTRACT.

The subtask and the task issuing the ATTACH macro-instruction are processed concurrently. Because both the task and its subtask compete for central processing unit time, control may pass from one to the other as program requirements and dispatching priorities dictate. Loading of the load module required by the subtask is done according to the subtask's priority. Loading is done asynchronously; therefore the creating task, even when it has a lower priority than the subtask, may continue to execute after execution of the ATTACH and before the subtask receives control.

Subtask Termination: When the subtask is terminated, the action taken depends on the operands in the ATTACH macro-instruction that created the subtask. These actions are described in Table 5.

Table 5. Supervisor Actions Upon Subtask Termination

Operands in ATTACH Macro-Instruction	Action by Supervisor at Subtask Termination
Neither ETXR nor ECB	The supervisor frees the subtask's resources and removes all control blocks (including the task control block) associated with the subtask. The supervisor does not inform the higher level task of its subtask's termination.
ECB only	The supervisor posts the event control block to indicate completion of the subtask, frees the subtask's resources, and removes all control blocks (except the task control block) associated with the subtask.
ETXR only	The supervisor frees the subtask's resources, removes all control blocks (except the task control block) associated with the subtask, and asynchronously enters the exit routine specified by the ETXR operand.
ECB and ETXR	The supervisor posts the event control block to indicate completion of the subtask, frees the subtask's resources, removes all control blocks (except the task control block) associated with the subtask, and asynchronously enters the exit routine specified by the ETXR operand.

Refer to Appendix C for additional information on the operation of ATTACH.

Upon entry to the exit routine specified by an ETXR operand, register contents are as follows:

Register	Contents
0	Internal supervisor information.
1	Address of the task control block for the subtask that terminated.
2-12	Same as when the interruption occurred.
13	Address of the supervisor-provided save area.
14	Return address (internal supervisor location).
15	Address of the exit routine. (This register can be used to provide addressability.)

Standard linkage conventions, including register saving and restoring responsibilities, apply.

Note that the ETXR routine is given control through an interruption of the task that issued the ATTACH macro-instruction.

The same ETXR exit routine can be specified each time a task creates a subtask, and it can be specified by more than one task. In these cases, the routine must be reusable, as follows:

- If the same ETXR routine is specified each time one task creates a subtask, the routine can be serially reusable (it need not be reenterable). When the subtasks terminate, the supervisor queues the terminations and gives control to the ETXR routine in a serial manner.
- If the same ETXR routine is specified by more than one task, the routine must be reenterable. The supervisor does not queue the terminations of two subtasks if the subtasks were created by different tasks.

The load module containing the ETXR routine must be in main storage when entry to the routine is required. Some of the ways the programmer can ensure that the routine will be in storage are as follows:

- If the ETXR routine is used by one task, it should be in the highest level load module used by the task.
- If the ETXR routine is used by more than one task, it should be in the highest level load module used by the highest level task of the job step.

Subpool Giving and Sharing: When the first task in a job step is created, only subpool 0 exists. The task can create additional subpools by means of GETMAIN macro-instructions. Any subpools the task creates can be given to, or shared with, subtasks that the task creates. Subpools, given to or shared with a subtask, can be given to, or shared with, subtasks that the subtask creates. A subpool that is given or shared is created, if it does not already exist, by execution of the ATTACH macro-instruction. The rules for giving and sharing subpools are:

1. Subpool zero.

   a. This subpool is shared by all tasks in a job step.

   b. No task or subtask can give ownership of subpool zero to a subtask.

   c. Specification of subpool zero in a GSPV, GSPL, SHSPV, or SHSPL operand is ignored.

2. Subpool ownership.

   a. Every subpool, except subpool zero, is owned by a task or subtask.

   b. Ownership is originally established by execution of a GETMAIN or ATTACH macro-instruction that creates a new subpool. A subpool created by the ATTACH macro-instruction is initially owned by the attaching task; ownership is retained if the subpool is shared with the subtask, and is transferred if the subpool is given to the subtask.

   c. Ownership can be given by the owning task to a subtask by a GSPV or GSPL operand in an ATTACH macro-instruction.

   d. After ownership is transferred, GETMAIN macro-instructions issued by the formerly owning task to request storage in the subpool create a new subpool with the same number.

3. Sharing subpools by SHSPV or SHSPL operands.

   a. If an owning task specifies the subpool in an SHSPV or SHSPL operand, the task and subtask share the subpool. The task retains ownership.

   b. If a sharing task specifies the subpool in an SHSPV or SHSPL operand, the task extends sharing of the subpool to its new subtask.

4. Giving of subpools by GSPV or GSPL operands

   a. If an owning task specifies the subpool in a GSPV or GSPL operand, the task loses ownership and the subtask is the new owner.

   b. If an owning task has shared a subpool, through SHSPV or SHSPL operands, with one or more subtasks, the task may not respecify the subpool in a GSPV or GSPL operand.

   c. If a sharing task specifies the subpool in a GSPV or GSPL operand, the task loses sharing of the subpool and the subtask receives sharing. Any other tasks sharing the subpool continue to share the subpool. Ownership of the subpool is unchanged.

5. Task termination.

   a. If the terminating task owns the subpool, even though it is sharing the subpool with other tasks, the storage occupied by the subpool is released for other uses. Therefore, the owning task should not terminate before all sharing tasks have stopped using the subpool.

   b. If the terminating task is sharing (but does not own) the subpool, the subpool is not affected.

6. GETMAIN and FREEMAIN macro-instructions.

   a. Any task that owns or shares a subpool can issue GETMAIN and FREEMAIN macro-instructions to extend or reduce the size of the subpool.

L- AND E-FORM USE: The standard form of this macro-instruction can result in a macro-expansion containing two parameter lists:

- A supervisor parameter list, which results from all operands of the macro-instruction except the PARAM and VL operands. This list is used by the supervisor to execute the macro-instruction.

- A problem program parameter list, which results from the optional PARAM operand. This list is identical to the parameter list resulting from a CALL macro-instruction, and is used to pass parameters to the load module being given control.

The standard-form macro-expansion can therefore consist of the following units of code:

Designation for Unit of Code And its Address	Code
AB	Branch to AE
ASPL	Supervisor parameter list
APL	Problem program parameter list
AE	Executable code terminated by an SVC instruction

When the ATTACH SVC instruction is executed, ASPL (the address of the supervisor parameter list) is in register 15, and APL (the address of the problem program parameter list) is in register 1.

Because both lists must be able to be remote, another special keyword operand, called the SF operand, is used in combination with the MF operand to provide nonstandard macro-instruction forms. The SF and MF operands can be written in the macro-instruction as shown in the following format:

$$\left\{\begin{array}{l} SF=L \\ SF=(E,\left\{\begin{array}{l}\text{spl-addrx}\\(15)\end{array}\right\}) \\ MF=(E,\left\{\begin{array}{l}\text{pl-addrx}\\(1)\end{array}\right\})[,SF=(E,\left\{\begin{array}{l}\text{spl-addrx}\\(15)\end{array}\right\})] \end{array}\right\}$$

In the above format, spl specifies the address of a remote supervisor parameter list, and pl specifies the address of a remote problem program parameter list. If (15) or (1) is written as shown, the address of the remote list must be loaded into the designated register before execution of the macro-instruction.

Four SF and MF combinations are shown in the above format. These result in macro-expansions consisting of the following units of code:

SF and MF Combination	Units of Code in Macro-Expansion
SF=L	ASPL
SF=(E,ASPL)	AB,APL,AE
MF=(E,APL)	AB,ASPL,AE
MF=(E,APL),SF=(E,ASPL)	AE

The effect of each SF and MF combination is as follows:

- SF=L results in only a supervisor parameter list. Neither the PARAM nor the VL operand can be written in the macro-instruction.
- SF=(E,ASPL) results in a macro-expansion that does not contain a supervisor parameter list. Parameters in the remote supervisor parameter list can be dynamically changed as in a normal E-form macro-instruction.
- MF=(E,APL) specifies a normal E-form macro-instruction. Note that MF=L cannot be written in the ATTACH macro-instruction, but a remote problem program parameter list can be formed by using the L form of the CALL macro-instruction.
- MF=(E,APL),SF=(E,ASPL) indicates that both parameter lists are remote.

DETACH -- Remove a Task (R)

The DETACH macro-instruction causes a specified subtask to be removed from the system. The DETACH frees the main storage area occupied by that subtask's task control block.

Name	Operation	Operand
[symbol]	DETACH	{tcbloc-addrx / (1)}

tcbloc
: specifies the address of a full-word whose three low-order bytes contain the address of the task control block for the subtask to be removed from the system.

  If (1) is written, the address of the full-word (not the address of the task control block) must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTIONS: A task's program can include DETACH macro-instructions for only those subtasks that were created by the task. A task must not attempt to detach the subtask of another task. A task must detach all of its subtasks before terminating.

ENVIRONMENT: If option 4 was excluded from the system, the DETACH macro-instruction is treated as a NOP.

EXCEPTIONAL RETURNS: If a task attempts to detach an incomplete subtask, the supervisor terminates the subtask and its subtasks abnormally. In this case, if asynchronous termination routines had been established by the subtask or any of its subtasks by means of STAE macro-instructions, those routines will be given control. The STAE exit

routines are executed in the same order as if an ABEND macro-instruction had been issued. However, the ETXR termination routines specified in ATTACH macro-instructions are not executed. This includes the routine that may have been specified in the ATTACH macro-instruction that created the subtask being terminated.

EXAMPLE: In the following example, the subtask associated with the task control block whose address is in the full-word at SAVTCBAD is to be removed from the system.

```
EX1     DETACH    SAVTCBAD
```

PROGRAMMING NOTES: When a task terminates, either normally or abnormally, most or all main storage areas related to the task are released. Storage release is handled in the same way for either a normal or abnormal termination.

If the task is the first in the job step, all storage areas are released, including the task control block areas, for the task and all of its subtasks.

If the task is a subtask, storage release is handled in one of the two following ways:

1. If neither an ECB nor an ETXR operand was specified in the ATTACH macro-instruction that created the subtask, all storage relating to the subtask and its subtasks is released.

2. If an ECB or an ETXR operand was specified, all storage, except the task control block area for the subtask being terminated, is released. It is for release of this task control block area that the DETACH macro-instruction is needed. Note that the task control block areas for the subtask's subtasks are released.

If a DETACH macro-instruction causes abnormal termination of a subtask, all areas belonging to the subtask are released.

CHAP -- Change Dispatching Priority (R)

The CHAP macro-instruction changes the dispatching priority of either the task issuing the macro-instruction or any of its subtasks. It may, in addition, increase the limit priority of a subtask by making the subtask's dispatching priority greater than its former limit priority.

Name	Operation	Operand
[symbol]	CHAP	{delta-value | (0)}, {tcbloc-{'S' | addrx} | (1)}

delta
: specifies the value to be algebraically added to the dispatching priority of the task specified by the second operand. Either a positive or a negative value can be specified. If this operand is written as an absolute expression, it can begin with a minus sign.

If (0) is written, the value must have been loaded into parameter register 0 before execution of this macro-instruction. If any register notation is written and the specified value is negative, the 32-bit two's complement of the absolute magnitude of the value must have been loaded into the designated register.

tcbloc
specifies the task whose priority is to be changed. If the dispatching priority of the issuing task is to be changed, 'S' should be written or the operand should be omitted. If the dispatching priority of one of the issuing task's subtasks is to be changed, either the addrx form or (1) should be written. Either form specifies the address of a full-word aligned on a full-word boundary; the three low-order bytes of the full-word should contain the address of the task control block (TCB) of the subtask.

If (1) is written, the address of the full-word (not the address of the task control block) must have been loaded into parameter register 1 before execution of this macro-instruction.

If (1) is written and parameter register 1 contains zero instead of an address, the dispatching priority of the task issuing the macro-instruction is changed.

CAUTION: If the CHAP macro-instruction is used to reduce the dispatching priority of the task issuing the macro-instruction, the task's dispatching priority may be made less than the dispatching priority of another task that is currently waiting for control. In this case, the second task will be given control immediately after execution of the CHAP.

ENVIRONMENT: If option 4 was excluded from the system, the CHAP macro-instruction is treated as a NOP.

EXAMPLES: In the following examples, EX1 increases the dispatching priority of the task issuing the macro-instruction by three. EX2 reduces by two the dispatching priority of the subtask whose task control block address is in location LTCBAD. EX3 either increases or decreases the dispatching priority of the task issuing the macro-instruction, depending on the contents of register 0 when the macro-instruction is executed. EX4 either increases or decreases (depending on the contents of register 0) the dispatching priority of the task whose task control block address is contained in the location specified in register 1 or, if register 1 contains zero, of the issuing task.

```
EX1  CHAP     3,'S'
EX2  CHAP     -2,LTCBAD
EX3  CHAP     (0),'S'
EX4  CHAP     (0),(1)
```

PROGRAMMING NOTES: A task cannot change its own limit priority. The dispatching priority of a task must be equal to its own limit priority or be between its limit priority and zero.

If a task attempts to make its own dispatching priority higher than its limit priority or below zero, its dispatching priority will be made equal to its limit priority or to zero, respectively. However, a task can increase the dispatching priority of its subtask up to the issuing task's own limit priority, even if this value exceeds the subtask's limit priority. In this case, the limit priority of the subtask is simultaneously raised to equal its new dispatching priority.

EXTRACT -- Extract Selected TCB Fields (S)

The EXTRACT macro-instruction provides the user's problem program with information contained in specified fields of the task control block (TCB) of either the task issuing the macro-instruction or one of its subtasks.

Name	Operation	Operand
[symbol]	EXTRACT	list-addr,[tcbloc-{'S' / addr}],FIELDS=({{ALL|GRS|FRS|TAX|AETX|PRI|CMC|TIOT},}...)

list
: specifies the address of a variable-length list of consecutive full-words of main storage. The list should be aligned on a full-word boundary. The supervisor will store the requested fields in this list. The number of full-words required equals the number of fields specified by the FIELDS operand.

tcbloc
: specifies the task control block whose fields are to be extracted. If the issuing task's task control block is to be specified, 'S' should be written or the operand should be omitted. If the task control block of a subtask is to be specified, the addr form should be written. The addr form specifies the address of a full-word, aligned on a full-word boundary; the three low-order bytes of the full word contain the address of the specified task control block.

FIELDS
: specifies, by a sublist of from one to seven values, the fields to be extracted:

Values	Fields to be Extracted
ALL	All of the following fields.
GRS	Address of the general register save area in which the supervisor saves the tasks's registers when the task is not in control. General registers are saved in the order 0 to 15.
FRS	Address of the floating-point register save area in which the supervisor saves the tasks's registers when the task is not in control.
TAX	Address of the entry point of the asynchronous, abnormal termination routine. (This routine is specified by means of the STAE macro-instruction.)
AETX	Address of the entry point of the asynchronous termination routine specified by the task that attached the task having this task control block. (This routine is specified by the ETXR operand of the ATTACH macro-instruction.)
PRI	Limit and dispatching priority values. (These values are stored into the third and fourth bytes, respectively, of the list word. The two high-order bytes of this word are set to zero.)
CMC	Task completion code. (If the task has not completed, this field is zero. Note that the completion code can be zero even if the task has completed.)
TIOT	Address of the task input/output table (TIOT).

Where the extracted field is an address, the address is stored in the three low-order bytes of the full-word in the list. The high-order byte of the word is set to zero.

If the TAX or AETX field is specified and the corresponding routine or interruption element does not exist, a zero address will be stored in the list.

The values can be written in the FIELDS operand in any order. However, the resulting list will contain the full-words in exactly the same order as shown above, and the list will be only as long as needed to hold the fields specified. (If a particular FIELD value is omitted, the corresponding full-word is omitted also.)

ENVIRONMENT: If option 4 was excluded from the system, the EXTRACT macro-instruction provides the address of the task input/output table only. The tcbloc operand is ignored, and the current task control block is assumed. The sublist for the FIELDS operand is searched for ALL or TIOT. If either is present the address of the task input/output table is inserted into the list. Other values in the sublist result in a full-word set to zero. Omitted values result in omitted words in the list.

EXAMPLES: In the following examples, EX1 results in three fields being extracted from the task control block of the task issuing the macro-instruction. The address of the general register save area is stored in the full-word at location LIST; the entry point address of the asynchronous termination routine, specified by the task that attached the issuing task, is stored in the full-word at LIST+4; the limit and dispatching priorities of the issuing task are stored in the full-word at LIST+8. Note that the order of the fields in the list follows the order shown in the description of the FIELDS operand, not the order in which FIELD values are written. The results of EX2 are exactly the same as those of EX1. EX3 results in all seven fields being extracted from the TCB whose address is in the full-word at location TCBADLC. These fields are stored in the seven full-words starting at location INFO.

```
EX1  EXTRACT   LIST,'S',FIELDS=(GRS,PRI,AETX)
EX2  EXTRACT   LIST,'S',FIELDS=(PRI,AETX,GRS)
EX3  EXTRACT   INFO,TCBADLC,FIELDS=(ALL)
```

L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B.

WAIT -- Wait for Event (R)

The WAIT macro-instruction specifies that the task issuing the macro-instruction should continue in control only when one or more specified events have occurred. An event can be the completion of another task; an operation requested by the current task, such as an input/output operation; or any other asynchronous operation. The completion of an event is indicated by execution of a POST macro-instruction.

Name	Operation	Operand
[symbol]	WAIT	[{count-value | (0)},] {ECB={addrx | (1)} | ECBLIST={addrx | (1)}}

count
: specifies the number of events that must occur before the task issuing the WAIT can continue in control. If the operand is omitted, one event is assumed.

  If (0) is written, the count must have been loaded into parameter register 0 before execution of this macro-instruction.

ECB
: specifies the address of an event control block (ECB) representing the only event that must occur before processing can continue. A description of an event control block is given in "Programming Notes" below.

  If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

ECBLIST
: specifies the address of a variable-length list. This list contains the addresses of up to 255 event control blocks, each of which represents an event to be waited for.

  Each ECB address is in the three low-order bytes of a full-word. These full-words must be consecutive and must be aligned on a full-word boundary. The sign bits in all but the last word in the list should be 0; the sign bit in the last word must be 1.

  If (1) is written, the list address must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTIONS: The value of the count operand must not exceed the number of event control blocks specified by the second operand.

Before its use by a WAIT macro-instruction, an ECB'S completion flag (bit 1) must be set to 0 by an instruction in the user's problem program. This must be done before the event represented by the ECB can occur. That is, the task that will subsequently issue a WAIT referring to an ECB must set the ECB's completion flag to 0 before it initiates the process that will result in posting of the ECB by another task or the control program. If the ECB is part of a data event control block

(DECB), as is the case in input/output operations, the control program sets the completion flag to 0 when a READ or WRITE macro-instruction is executed.

The program must not issue a WAIT macro-instruction that refers to an ECB whose wait flag is set to 1. Note that execution of a WAIT macro-instruction always terminates with the wait flags set to 0 in all event control blocks referred to by the macro-instruction. The responsibility of the user's problem program is merely to ensure that it itself does not inadvertently set a wait flag to 1.

The same ECB must not be used to represent an event that will be waited for by more than one task or asynchronous exit routine at one time. This is because an ECB must not be referred to by a WAIT macro-instruction when the wait flag in the ECB is already set to 1.

WAIT and POST macro-instructions used to synchronize two tasks or the synchronous and asynchronous processing of one task can result in permanent interlock. For example, a task could issue a WAIT macro-instruction for completion of a subtask. If the subtask contained a WAIT macro-instruction for an event that can occur in the task only after the task's WAIT macro-instruction is satisfied, both the task and the subtask enter permanent wait conditions.

EXCEPTIONAL RETURNS: Control will be returned to a task before the waited event when necessary to allow execution of an asynchronous exit routine. The routine may be part of the operating system or of the user's problem program, and may be any of the following:

- An input/output or external interruption routine.
- A subtask termination exit routine, specified by the ETXR operand of an ATTACH macro-instruction previously executed by the task.
- A task abnormal exit routine, specified by a STAE macro-instruction previously executed by the task.

Except in the last case (task termination), the task is returned to the wait condition when execution of the exit routine is complete.

ENVIRONMENT: If options 1, 2, and 4 have been excluded from the system, only a single event can be waited for. A non-zero count operand is ignored and a single event is assumed. The ECBLIST operand must not be used.

A load module can wait only for the completion of events that it initiates or are initiated by a load module that it has attached.

EXAMPLES: In the following examples, EX1 results in a wait for a single event represented by the ECB at location ALPHA. EX2 results in a wait for a single event represented by any of the event control blocks pointed to by the list of addresses located at the address contained in register 1. Because the count operand is omitted, a count of 1 is assumed.

```
EX1  WAIT    1,ECB=ALPHA
EX2  WAIT    ECBLIST=(1)
```

PROGRAMMING NOTES: Execution of a WAIT macro-instruction indicates to the supervisor that the issuing task cannot continue in control unless one or more specified events have occurred. The supervisor determines whether an event has occurred by examining an event control block specified by the WAIT macro-instruction.

An ECB is a full-word of main storage provided by the user's problem program to represent an event. It must be aligned on a full-word boundary, and has the format shown in Figure 3.

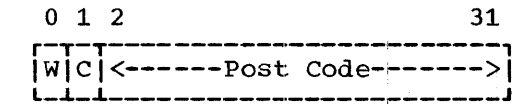


Figure 3. Format of the Event Control Block (ECB)

W - Wait Flag: The wait flag is set to 1 by the supervisor when the ECB is referred to by a WAIT macro-instruction and the event has not occurred. The supervisor subsequently sets the wait flag to 0 when either of the following occurs:

- When the supervisor sets the completion flag to 1 (because the ECB has been referred to by a POST macro-instruction).
- When execution of the WAIT macro-instruction is complete because a specified subset of events has occurred. In this case, the supervisor sets to 0 the wait flags in the event control blocks that represent events that have not yet occurred.

C - Completion Flag: The completion flag is set to 1 by the supervisor upon execution of a POST macro-instruction that refers to the ECB and indicates completion of the event. This action occurs whether or not the ECB is referred to by a WAIT macro-instruction. The user's problem program must subsequently set the completion flag to 0, if it intends to reuse the ECB to represent another event, except when the ECB is part of a data event control block (DECB).

Post Code: This field contains information specified by the code operand of a POST macro-instruction, the compcode operand of an ABEND macro-instruction, or the code operand of a RETURN macro-instruction. (The ABEND and RETURN codes are placed in an event control block only if the ATTACH macro-instruction that created the terminating task contained an ECB operand.) The supervisor stores the code into the ECB when it sets the completion flag to 1. This action occurs upon execution of the POST, ABEND, or RETURN macro-instruction.

If a code was not written in the POST, ABEND, or RETURN macro-instruction, the post code field is set to zero upon execution of one of these macro-instructions.

As long as the wait flag is set to 1, the post code field contains infomation required by the supervisor. This information indicates that the task that issued the WAIT macro-instruction is currently waiting for the represented event to occur. The user's problem program must not alter the post code field when the wait flag is 1.

Event control blocks and the actions of the WAIT, POST, ABEND, and RETURN macro-instructions are the principal means by which program execution that depends on asynchronously occurring events is achieved. Therefore, the address of an ECB usually appears as an operand of a system macro-instruction that results in an asynchronous operation.

For example, assume that task A executes two ATTACH macro-instructions, each addressing a different ECB. Also assume that both subtasks must be complete before task A can proceed beyond a point, X.

At point X, task A should have a WAIT macro-instruction with a count operand of two, and an ECBLIST operand specifying the two event control blocks addressed by the ATTACH macro-instructions.

Assume that one subtask is completed before the WAIT macro-instruction is executed. The supervisor posts the corresponding ECB and sets its completion flag to 1. If the WAIT macro-instruction is now executed, the wait flag in the ECB corresponding to the unfinished subtask is set to 1, and task A is placed in a wait condition. The address of task A's task control block (or an equivalent control block internal to the supervisor) is placed in the post code field of the ECB whose wait flag is set to 1. (Because this field can contain only one such address at one time, an ECB cannot be referred to simultaneously by two or more WAIT macro-instructions.) A count is formed in a control block internal to the supervisor; this "wait count" indicates the number of events for which task A is waiting.

When the second subtask is completed, the wait flag is set to 0 and the completion flag is set to 1 in the corresponding ECB. The wait count is reduced by one, and, because there are no more events to be waited for, task A is removed from the wait condition and is able to compete for control. This process requires use of the address previously stored in the post code field.

If a task issues a WAIT macro-instruction when the completion flags in all of the specified event control blocks are 1, the task remains in control.

When a WAIT macro-instruction is satisfied, the issuing task can interrogate the completion flags and post codes in the specified event control blocks to determine which events occurred. By post codes agreed to between the tasks issuing the WAIT and POST, ABEND, or RETURN macro-instructions, the task issuing the WAIT can determine how each event was completed.

If a WAIT macro-instruction has a count operand that is less than the number of specified event control blocks, the WAIT will be satisfied by a subset of the events. At this point, the supervisor will reset the wait flags of all remaining event control blocks to 0. Subsequent completion of the remaining events will result in unpredictable posting of the corresponding event control blocks. Therefore, care should be exercised when one ECB is referred to by two or more WAIT macro-instructions.

A count of zero results in a NOP.

WAITR -- Wait for Event and Ready Lower Priority Task (R)

The WAITR macro-instruction is similar in function and format to the WAIT macro-instruction. WAITR is used to initiate the execution of the lower priority task in a system in which option was included. This occurs upon the first execution of WAITR by the higher priority task.

WAITR is written as shown in the WAIT macro-instruction description.

ENVIRONMENT: If option 2 was excluded from the system, WAITR functions in the same way as WAIT.

Before the first execution of WAITR, job steps are executed, as the higher priority task, in one of the two partitions. WAITR should be issued only after termination of the first input stream has occurred. The last job in the input stream may read data from the input stream

until it issues the WAITR macro-instruction. The instruction readies the lower priority task and a message from the control program to the operator will request that a second input stream be started. The job step that issued the WAITR continues to execute, as the higher priority task, in the first partition. Subsequently initiated job steps are executed, as the lower priority task, in the second partition.

Both WAITR and WAIT, issued by the higher priority task, cause control to be given to the lower priority task until the specified events have occurred.

PROGRAMMING NOTES: A WAITR macro-instruction issued by a lower priority task functions in the same way as the WAIT macro-instruction.

With option 2, if the higher priority task issues a WAITR having a count operand whose value is zero, the request is recognized as a WAIT macro-instruction and is treated as a NOP; a wait condition and task interchange in main storage do not occur.

POST -- Signal Event Completion (R)

The POST macro-instruction signals that the event represented by a specified event control block (ECB) has occurred. Posting of an ECB is used to satisfy the requirements of a WAIT macro-instruction.

Name	Operation	Operand
[symbol]	POST	{ ecb-addrx / (1) } , { code-value / (0) }

ecb
: specifies the address of an ECB representing an event whose completion is signaled by execution of this macro-instruction.

  If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

code
: specifies a value to be placed in the post code field of the addressed ECB. This value should be from 0 to $2^{30}-1$. If the operand is omitted, a post code of zero is assumed.

  If (0) is written, the value must have been loaded into parameter register 0 before execution of this macro-instruction.

CAUTION: The completion flag (bit 1) in the specified ECB should be 0 when the POST macro-instruction is issued. If it is not, the posting will result in no action.

EXAMPLES: In the following examples, EX1 posts the event control block ECBAA and sets its post code to zero. EX2 posts the event control block whose address is in register 5, and sets its post code with the value contained in register 6.

```
EX1  POST     ECBAA
EX2  POST     (5),(6)
```

PROGRAMMING NOTES: Refer to the WAIT macro-instruction's "Programming Notes."

ENQ -- Enqueue Request for a Serially Reusable Resource (R)

The ENQ macro-instruction is used by a task to place itself on a first-in, first-out queue in order to gain access to a serially reusable resource, such as a program (one that may be entered by a direct linkage by this or another task) or a data area. All tasks that require use of the resource should request its use by issuing ENQ macro-instructions that refer to the same queue. The queue is formed by means of a queue control block whose address is known by all tasks that request the resource. A task is represented on the queue by a queue element. Until a task's queue element reaches the top of the queue, the task is held in a wait condition. When the task's queue element reaches the top of the queue, the task again competes for control and can use the resource.

A DEQ macro-instruction is used to remove a task's queue element from the top of the queue. It should be issued by a task when the task is finished using the serially reusable resource.

Name	Operation	Operand
[symbol]	ENQ	{qcb-addrx / (1)},{qel-addrx / (0)}

qcb
: specifies the address of a queue control block (QCB) that corresponds to the serially reusable resource. A description of a queue control block is given in "Programming Notes" below.

  If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

qel
: specifies the address of an eight-byte queue element representing the task. The queue element is eight bytes long and must be aligned on a full-word boundary.

  If (0) is written, the element address must have been loaded into parameter register 0 before execution of this macro-instruction.

CAUTIONS: The queue pointer of a QCB must be 0 when the corresponding resource is free and the QCB is referred to by an ENQ macro-instruction. If it is not, the current task, and all other tasks referring to the same QCB, will be placed permanently in a wait condition, since no DEQ macro-instruction can be issued.

Because the time during which asynchronous interruptions occur cannot be predicted, two or more tasks should not directly manipulate the queue pointer of a QCB. The tasks should use the ENQ and DEQ macro-instructions.

ENVIRONMENT: If option 4 was excluded from the system, the ENQ macro-instruction is treated as a NOP.

EXAMPLES: In the following examples, EX1 queues its task on the queue control block named AREA5. The queue element, whose address is in register 0, is presented to the control program for enqueuing on AREA5

QCB. If no other queue elements are on the queue, the task issuing the ENQ macro-instruction resumes control; otherwise, the task will be placed in a wait condition until its queue element reaches the top of the queue. EX2 makes a request using the QCB whose address is AREA5 plus the contents of register 5. The queue element is at location MINE.

```
EX1  ENQ      AREA5,(0)
EX2  ENQ      AREA5(5),MINE
```

PROGRAMMING NOTES: Tasks can control the use of a serially reusable resource by using the ENQ and DEQ macro-instructions, a QCB corresponding to the resource, and queue elements representing the competing tasks. The QCB and the queue elements are in main storage owned by the tasks involved. The address of the QCB is known by all the tasks.

A QCB is a full-word of main storage provided by the user's problem program. It corresponds to a specific program, data area, or other resource. The QCB is aligned on a full-word boundary and has the format shown in Figure 4.

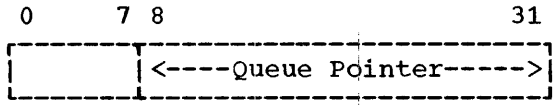


Figure 4. Format of the Queue Control Block (QCB)

Bits 0 through 7: Reserved for use by the supervisor.

Queue Pointer: The queue pointer is the address of the first of a chain of queue elements.

Each queue element is provided by a unique task that has issued an ENQ macro-instruction, and has not subsequently issued a DEQ macro-instruction. The first queue element, the one addressed by the queue pointer, represents the task that is currently in competition for control and has exclusive use of the resource. This task is called the owning task. The remainder of the queue elements represent tasks in wait conditions.

When the first task issues an ENQ macro-instruction for a resource, the request is immediately satisfied, and the task continues in control. The supervisor stores in the queue pointer field of the QCB the address of the queue element provided by the task.

If a second task issues an ENQ macro-instruction for the same resource before the the owning task issues a DEQ macro-instruction, the second task is placed in a wait condition. The supervisor enqueues the queue element of the second task by storing its address in the queue element of the first task.

When the first task releases the resource by issuing a DEQ macro-instruction, the task's queue element is removed from the queue. The queue pointer field in the QCB is updated to point to the next element in the queue, and the task represented by that element is removed from wait condition. That task can now compete normally for control.

After a queue element has been removed from the queue by execution of a DEQ macro-instruction, the eight bytes of the element can be used in any way by the user's problem program.

When the last queue element is removed from the queue, the supervisor sets the queue pointer in the QCB to 0.

ENQ macro-instructions in a task and one of its asynchronous exit routines must not specify the same QCB. For example, the task's program could issue an ENQ macro-instruction and, if it is first on the queue, continue processing. Then, through an interruption, an asynchronous exit routine could be given control. If the routine issued an ENQ macro-instruction specifying the same QCB, its queue element would be placed second on the queue. Now, both the task's program and the exit routine are in a permanent wait condition: the task is waiting for return of control from the exit routine and the exit routine is waiting for the task to issue a DEQ macro-instruction.

DEQ -- Dequeue Request for a Serially Reusable Resource (R)

The DEQ macro-instruction is used by a task to remove itself from the top of a first-in, first-out queue that was used by the task to gain access to a serially reusable resource. The queue was formed by means of a queue control block whose address is known by all tasks that use the resource. The task is represented on the queue by a queue element that was placed on the queue when the task issued an ENQ macro-instruction. The DEQ macro-instruction should be issued only when the task is finished using the serially reusable resource. This releases the resource for use by the task whose queue element is next on the queue.

Name	Operation	Operand
[symbol]	DEQ	{qcb-addrx / (1)}

qcb
: specifies the address of a queue control block (QCB) corresponding to the serially reusable resource.

  If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

The address of the eight-byte queue element representing the task issuing the DEQ macro-instruction is returned in register 0.

ENVIRONMENT: If option 4 was excluded from the system, the DEQ macro-instruction is treated as a NOP.

EXAMPLES: In the following examples, EX1 removes its task's queue element from the queue under the queue control block at location AREA5. The address of the queue element that was enqueued on the AREA5 QCB is returned to the user's program in register 0. EX2 removes its task's queue element from the queue under the QCB at the indicated explicit address. Again, the address of the now-unused queue element is returned in register 0.

```
EX1  DEQ      AREA5
EX2  DEQ      0(7,10)
```

PROGRAMMING NOTES: Refer to the ENQ macro-instruction's "Programming Notes."

SPIE -- Specify Program Interruption Exit (S)

The SPIE macro-instruction specifies the address of an exit routine to be entered asynchronously when specified program interruptions occur in the task issuing the SPIE macro-instruction. The effect of each SPIE macro-instruction issued by a task supersedes the effect of the SPIE macro-instruction issued previously by the same task. The program mask in the program status word (PSW) is set as specified by this macro-instruction.

Name	Operation	Operand
[symbol]	SPIE	[exit-addr,interrupts-({x|(y,z)},...)]

exit
specifies the address of the exit routine.

interrupts
specifies the program interruption types that are to cause the exit routine to be entered. Each operand of the sublist is one of the following:

1. A single decimal number, to indicate one program interruption type. (See x in the format description.)

2. A pair of decimal numbers, to indicate a continuous group of program interruption types. The numbers must be separated by a comma and enclosed in parentheses. The second number must be greater than the first. (See (y,z) in the format description.)

The interrupts values can be written in any order.

Each number must be from 1 through 15. The numbers indicate the following program interruption types:

Number	Interruption Type	Able to be Masked
1	Operation	No
2	Privileged operation	No
3	Execute	No
4	Protection	No
5	Addressing	No
6	Specification	No
7	Data	No
8	Fixed-point overflow	Yes
9	Fixed-point divide	No
10	Decimal overflow	Yes
11	Decimal divide	No
12	Exponent overflow	No
13	Exponent underflow	Yes
14	Significance	Yes
15	Floating-point divide	No

If a maskable interruption type is specified by this operand, the corresponding program mask bit in the PSW is set to 1 by the supervisor. This enables the interruption. The program mask bits

for all unspecified maskable interruption types are set to 0 so that the interruptions cannot occur.

Before the first SPIE macro-instruction is executed by a task, all maskable program interruptions are masked so that they will not cause an interruption, and all nonmaskable interruptions cause the standard system exit routine to be given control. These conditions also exist after a SPIE macro-instruction in which both operands are omitted is executed.

After execution of the first SPIE macro-instruction by a task, register 1 contains zero. After execution of a succeeding SPIE macro-instruction by the same task, register 1 contains the address of the program interruption control area (PICA) (see Figure 5) formed for the preceding SPIE macro-instruction. This address can be used subsequently to restore the superseded interruption monitoring conditions. (Refer to "L- and E-Form Use" below.)

CAUTIONS: If a called program uses the SPIE macro-instruction or otherwise changes the program mask, it must restore the program interruption control area and the program mask to their original status before returning to the calling program. (Refer to "Linkage Conventions" in Section 1.)

A SPIE macro-instruction that specifies a maskable interruption type, changes the program mask; thus, it overrides a previous SPM instruction that was executed directly by the user's problem program.

ENVIRONMENT: If option 4 was excluded from the system, SPIE specifies the exit routine and interruption monitoring conditions for the job step and any module that may be attached. If SPIE is issued by an attached module, the specified exit routine supersedes the routine previously established for the job step.

EXAMPLES: In the following examples, EX1 specifies that if an operation, fixed-point overflow, or floating-point divide interruption occurs, control is to be passed to location FIXUP. When this macro-instruction is executed, the supervisor sets the program mask in the PSW so that a fixed-point overflow exception causes an interruption, while the other maskable interruption types (decimal overflow, exponent underflow, and significance exceptions) do not cause an interruption.

EX2 specifies that program interruption types 1 through 7 and 10 through 15 will cause the routine at ERRTN to be entered.

EX3 indicates that the task no longer desires to give special treatment to program interruptions and does not want maskable interruptions to occur.

```
EX1     SPIE    FIXUP,(1,8,15)
EX2     SPIE    ERRTN,((1,6),(10,15),7)
EX3     SPIE
```

PROGRAMMING NOTES: This macro-instruction causes a program interruption control area to be created at assembly time. This area is aligned on a full-word boundary, is six bytes long, and has the format shown in Figure 5.

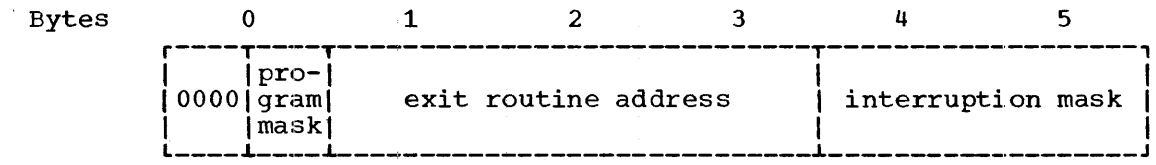


Figure 5. Format of the Program Interruption Control Area (PICA)

The four high-order bits of byte 0 are zeros; the four low-order bits of byte 0 correspond to the program mask (bits 36 through 39 of the program status word). Bytes 1, 2, and 3 contain the address of the exit routine. Bytes 4 and 5 contain a mask resulting from the interrupts operand. In this mask, the bits are numbered 0 through 15; specification of an interruption type causes the corresponding bit to be set to one. Bit zero of the mask is reserved for use by the supervisor. If both operands were omitted from the macro-instruction, all six bytes would contain zeros.

On the first execution of a SPIE macro-instruction by a task, the supervisor forms a 32-byte program interruption element (PIE) aligned on a double-word boundary in subpool zero. (Subpool zero is shared by all tasks of a job step.) The format of the program interruption element is shown in Figure 6.

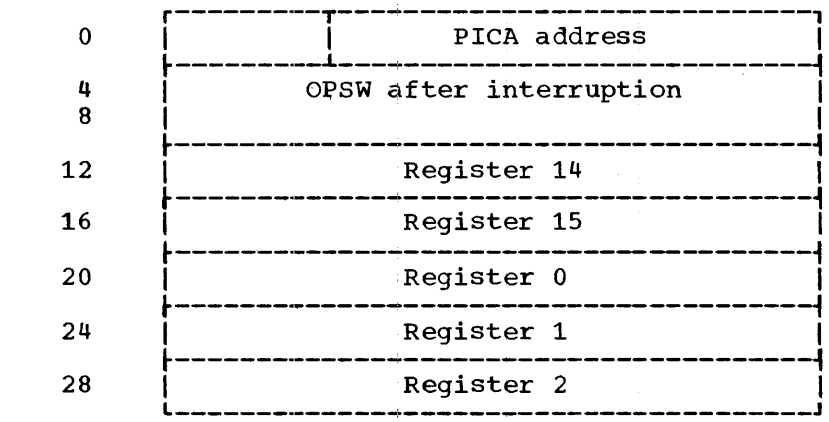


Figure 6. Format of the Program Interruption Element (PIE)

Byte 0 is reserved for use by the supervisor. Bytes 1, 2, and 3 contain the address of the program interruption control area resulting from the SPIE macro-instruction most recently executed by the task. The remaining bytes contain information placed in them by the supervisor upon program interruption.

Note that the program interruption control area address in the program interruption element changes each time another SPIE macro-instruction is executed by the task.

The previous program interruption control area address is returned in register 1 to the task issuing the macro-instruction. This allows a subprogram to specify its own program interruption handling procedures, and to then use the returned address to restore the procedures specified by the previous subprogram. This process is described in "L- and E-Form Use" below.

If a program interruption element does not exist when a program interruption occurs, the supervisor gives control to the standard system exit routine. Otherwise, the supervisor stores the contents of the old program status word (OPSW) in bytes 4 through 11 of the program interruption element, and the contents of registers 14, 15, 0, 1, and 2 in bytes 12 through 31. The supervisor then examines the interruption mask in the task's current program interruption control area. If the bit corresponding to the current interruption is 1, control is given to the exit routine whose address is in the program interruption control area. If the bit is 0, control is given to the standard system exit routine.

If a program interruption occurs when the exit routine specified by the program interruption control area is in use by this task, the standard system exit routine is given control.

Note that if a mask bit in the program interruption control area is 0 and the bit corresponds to a maskable program interruption, the corresponding bit in the program mask in the program status word will normally be 0 also, and that type of interruption will not occur. However, the task could have executed a SPM instruction to set the program mask bit to 1. This would have been done if the task desired the exceptional condition to be recognized, and if the resulting interruption was to give control to the standard system exit routine.

Upon entry to the exit routine specified in a SPIE macro-instruction, register contents are:

Register	Contents
0	Internal supervisor information.
1	Address of the program interruption element for the task that caused the interruption.
2-13	Same as when the interruption occurred.
14	Return address (internal supervisor location).
15	Address of the exit routine. (This register can be used to provide addressability.)

Note that the program interruption element contains the contents of registers 14, 15, 0, 1, and 2 as they were when the interruption occurred.

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

Upon completion of the exit routine, the contents of register 14 must be as they were upon entry to the routine. The exit routine should terminate with a branch to the address in register 14.

The exit routine can alter the contents that will be in 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, because these registers are reloaded from this area by the supervisor when it returns control to the interrupted program. The 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.

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

One SPIE exit routine can be used by more than one task of a job step. If the routine is used in this manner, it must be reenterable.

L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B.

When a SPIE macro-instruction is executed, the address of the previously specified program interruption control area is returned in register 1.

This area can be reinstated as the current program interruption control area by being designated as the remote parameter list used in an E-form macro-instruction. For example, if the address returned in register 1 were saved in register 9, the following macro-instruction could be issued:

```
SPIE   MF=(E,(9))
```

STAE -- Specify Task Abnormal Exit (R)

The STAE macro-instruction specifies the address of an exit routine to be entered asynchronously if the task issuing the STAE macro-instruction terminates abnormally.

The effect of each STAE macro-instruction issued by a task supersedes the effect of the STAE macro-instruction issued previously by the same task.

Name	Operation	Operand
[symbol	STAE	{exit-addrx} {(1)}

exit
: specifies the address of an exit routine to be entered if the task issuing this macro-instruction terminates abnormally.

  If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

  If register notation is used and the designated register has been loaded with zero, no exit routine is specified, and any previous execution of a STAE macro-instruction in the same task is ignored. That is, a STAE exit routine will no longer be executed upon abnormal termination.

The task issuing the STAE terminates abnormally when one of the following occurs:

- The task issues an ABEND macro-instruction.
- A higher level task is terminated.
- Any task in the job step issues an ABEND with a STEP operand.
- The control program issues an ABEND for the task.

ENVIRONMENT: If option 4 was excluded from the system, the STAE macro-instruction is treated as a NOP.

EXAMPLES: In the following examples, EX1 specifies that, if the task terminates abnormally, the routine at PMDUMP is to be entered.

When EX2 is executed, register 1 can contain either the address of an exit routine or zero. In the latter case, any STAE macro-instruction previously executed by the same task will be ignored.

```
EX1     STAE    PMDUMP
EX2     STAE    (1)
```

PROGRAMMING NOTES: The specified exit routine must be in main storage when its task terminates abnormally. Therefore, the routine usually should be in the highest level load module of the task issuing the STAE macro-instruction.

Upon entry to the exit routine, the general registers will contain:

Register	Contents
0	Internal supervisor information.
1	Abnormal completion code.
2-12	Not predictable.
13	Address of the supervisor-provided save area.
14	Return address (internal supervisor location).
15	Address of the exit routine. (This register can be used to provide addressability.)

Standard linkage conventions, including register saving and restoring responsibilities, apply.

One STAE exit routine can be used by more than one task of a job step. If the routine is used in this manner, it must be reenterable.

ABEND -- Terminate a Task Abnormally (R)

The ABEND macro-instruction causes abnormal termination of the task issuing the macro-instruction; it also causes all of the task's incomplete subtasks to be terminated abnormally. The ABEND macro-instruction can optionally cause abnormal termination of all incomplete tasks in the job step, followed by termination of the job.

Name	Operation	Operand
[symbol]	ABEND	{compcode-value | (1)},[DUMP][,STEP]

compcode
: specifies a completion code to be stored in the task completion code field of the task control block (TCB) of the task issuing the ABEND. If an ECB operand appeared in the ATTACH macro-instruction that created the current task, the completion code is also stored in bits 8 through 31 of the post code field of the specified event control block. Bits 2 through 7 of the post code are set to zero.

  If this macro-instruction results in termination of the job step, the completion code is recorded on the system output.

  The value of this operand should be a multiple of 4 and must be less than 4096. (Refer to "Use of ABEND by Control Program.")

  If (1) is written, the completion code must have been loaded into parameter register 1 before execution of this macro-instruction.

DUMP

specifies that an abnormal termination dump is to be taken. This dump is taken only if the user has specified a data set by means of a DD control statement whose ddname is SYSABEND. It includes the contents of registers and main storage areas occupied by the programs, data, and system control blocks for all tasks being abnormally terminated.

STEP

specifies that all tasks in the job step are to be terminated, and that subsequent steps of the same job are to be ignored. If this operand is omitted, only the task issuing the ABEND macro-instruction and its incomplete subtasks are terminated.

ENVIRONMENT: If option 4 was excluded from the system, execution of an ABEND macro-instruction results in abnormal termination of the current job step. The STEP operand is ignored. The completion code is stored in the task control block of the job step, not in the event control block specified by the ATTACH macro-instruction. Abnormal terminations are not posted.

If the DUMP operand is present, an indicative dump may be taken in place of a complete abnormal termination dump. An indicative dump is taken when the SYSABEND DD control statement has been omitted or the corresponding entry in the task input/output table (TIOT) has been destroyed. It consists of a limited amount of information about the contents of registers and system tables, and is printed automatically on the system output device.

Asynchronous exits (of the type specified in STAE and ATTACH) are not provided upon abnormal termination.

If there is not sufficient main storage to accommodate the ABEND procedure, storage allocated to the job step is used. This allocation may mean that data sets cannot be closed and may alter the dump.

EXAMPLES: In the following examples, EX1 causes abnormal termination of the task issuing the ABEND and all of its incomplete subtasks. The value, 256, is placed in the task completion code field of the task control block of the current task. If an ECB operand was specified in the ATTACH macro-instruction that created the current task, the value is placed in the event control block associated with the current task.

EX2 causes abnormal termination of the job step. The value, 12, is recorded on the system output. All of the job step's main storage areas are recorded on external storage.

```
EX1     ABEND   256
EX2     ABEND   12,DUMP,STEP
```

PROGRAMMING NOTES: The ABEND macro-instruction abnormally terminates the current job step if it is executed under the highest level task, or if the STEP operand has been specified. Termination of the job step causes the completion code for the highest level task to be printed on the system output device; it also causes the job to be terminated and subsequent job steps to be left unexecuted.

Abnormal Termination Dump: The data control block for an abnormal termination dump is provided by the control program and is opened automatically. Its DDNAME parameter refers to a DD control statement named SYSABEND, which must be supplied by the programmer. The parameters of the DD control statement determine whether the data set is printed immediately or stored for later printing by a utility program.

For immediate printing of a dump, the UNIT parameter of the SYSABEND DD control statement should specify a printer. For automatic processing of the dump, the SYSABEND DD statement should specify SYSOUT.

For storage of the dump, the DISP parameter should specify KEEP or CATLG, and the UNIT parameter should specify an external storage device, such as a magnetic tape unit or direct-access device. The VOLUME parameter should specify a single volume. The SPACE operand should specify sufficient space (when a direct-access device is employed) to contain the dumped information. If (1) option 4 is excluded from the system and (2) the primary quantity reserved is insufficient and (3) sufficient main storage is not available to extend the data set, then the dump will not be completed.

Asynchronous Exit Routines: Upon abnormal termination by an ABEND macro-instruction, control is always given to the STAE macro-instruction exit routine of any task that is terminated. Control is given to the ETXR exit routine (specified in an ATTACH macro-instruction) of a task only if its immediate subtask issued the ABEND macro-instruction, and the STEP operand was omitted.

The STAE exit routine is specified through a STAE macro-instruction previously executed by a task being terminated. The ETXR exit routine is specified by an ETXR operand in the ATTACH macro-instruction that created the task being terminated.

Tasks terminate abnormally either directly or indirectly, as follows:

1. The highest level task of a job step terminates directly when one of the following occurs:

   a. The task issues an ABEND macro-instruction.
   b. Any task of the job step issues an ABEND macro-instruction with the STEP operand.

2. A task other than the highest level task of a job step terminates directly when it issues an ABEND macro-instruction without the STEP operand.

3. A task terminates indirectly when an ABEND macro-instruction that will cause the next higher level task to terminate is issued.

When a task is to be terminated, its subtasks are terminated first, whether the termination is direct or indirect. If a task and all its subtasks are to be terminated and the subtasks are at different control levels, the lowest control level subtask is terminated first. The immediate subtasks of one task can be terminated in any order.

When a task (or subtask) terminates (either directly or indirectly), control is given to its STAE routine. Upon return of control from the STAE routine to the control program, the control program posts the event control block (specified by an ECB operand in the ATTACH macro-instruction that created the terminating task) associated with termination of the task.

Finally, if the terminating task terminates directly, control is given to the ETXR routine (if one was specified) of the task that created the terminating task. Control is not given to the ETXR routine of a task when its subtask terminates indirectly.

Use of ABEND by Control Program: When the control program detects an error condition requiring termination of a task, the control program issues an ABEND macro-instruction for the task. The completion code operand value will be from 4096 to $2^{24}-1$ in multiples of 4096.

Whenever the ABEND macro-instruction is used, the completion code consists of 24 bits. All 24 bits are placed in the task completion code field in the task control block and in the post code field in the event control block. The 12 high-order bits are reserved for the control program, which uses them to indicate the reason for the abnormal termination of the task.

The control program will output an abnormal completion code in each of the following cases:

- When any task is terminated, and the user provided the SYSABEND DD statement.
- When the highest level task of the job step is terminated, whether or not the user provided the SYSABEND DD statement.

After the control program has abnormally terminated a task, a higher level task's program can interrogate the bits set by the control program.

CHKPT -- Checkpoint a Job Step (R)

The CHKPT macro-instruction causes the programs, main storage data areas, and system control blocks for a job step to be recorded on external storage. After the checkpoint has been taken (onto a checkpoint data set), execution of the task that issued the CHKPT macro-instruction continues with the next sequential instruction. Execution of the job step can be reinitiated from the last checkpoint by executing a restart procedure through the job stream.

The job step issuing a CHKPT macro-instruction may use the execute channel program mode for data sets other than the checkpoint data set; if this is done, the user's problem program should ensure that the block counts in the data control blocks for these data sets are valid when the CHKPT is issued. Incorrect block counts will result in incorrect repositioning of the data sets upon restart.

ENVIRONMENT: If option 4 was included in the system, a request for a checkpoint is accepted only if the other tasks in the step have been completed and the step is in a single task state.

When this macro-instruction is executed, all outstanding input/output activities in the job step are allowed to complete. If the data control block for the user's checkpoint data set has not been previously opened, the supervisor opens it. The supervisor then records the main storage areas containing the programs, data, and system control blocks for the job step. After the checkpoint is taken, the supervisor returns control to the location that immediately follows the CHKPT macro-instruction.

All checkpoints in a job step can refer to the same data control block and can therefore be written on the same data set; or, the checkpoints can refer to different data control blocks and be written on different data sets.

The maximum number of checkpoints that can be written by a job step on a nonremovable direct-access device is either $2^{16}-1$ or the number that fills the volume, whichever is less. However, checkpoints can be written on any number of devices, so that this limitation applies only to any one device. Each device requires a different data control block.

The maximum number of checkpoints that can be written on a multivolume data set on a device with removable volumes is $2^{16}-1$. When a volume is filled, the supervisor will order the operator to replace the volume so that more checkpoints can be written.

For each checkpoint data control block, checkpoints are numbered from 1 to $2^{16}-1$ in the order in which they are taken. However, the restart procedure can be used to initiate the job step only from the last checkpoint written on the corresponding data set. If checkpoints were written on more than one data set, the restart procedure must specify the correct data control block for the data set.

A task's data sets are not saved on the checkpoint data set and, therefore, cannot be restored. Restarts are meaningful only if data is available and unchanged after the checkpoint was taken.

TIME -- Request Time and Date (R)

The TIME macro-instruction provides the time of day in register 0 and the current date in register 1.

Name	Operation	Operand
[symbol]	TIME	[unit-{DEC|BIN|TU}]

unit
specifies the units in which the time of day is to be returned in register 0. The values for this operand and their meanings are:

DEC specifies that the time is to be returned as eight packed decimal digits in the format HHMMSSth, where HH = hours in a 24-hour clock, MM = minutes, SS = seconds, t = tenths of seconds, and h = hundredths of seconds.

BIN specifies that the time is to be returned as an unsigned 32-bit binary number in which the least significant bit has a value of 0.01 second.

TU specifies that the time is to be returned as an unsigned 32-bit binary number in which the least significant bit has a value of 1 timer unit. A timer unit is 26 microseconds.

If this operand is omitted, DEC is assumed.

The time returned is the time of day based on a 24-hour clock that is set with real local time by the operator.

The date is always returned in register 1 in five packed decimal digits in the format YYDDD, where YY = the last two digits of the year and DDD = the day of the year. These five digits are preceded in register 1 by two packed decimal zeros and followed by a four-bit character. Because of the bit pattern of this character, all digits will have the same zone when the 32-bit field is unpacked.

ENVIRONMENT: If both option 6A and 6B were excluded from the system, only the date is provided in register one. The unit operand is ignored.

EXAMPLES: In the following examples, EX1 requests the time of day in eight packed decimal digits (the DEC format). EX2 requests the time of day in timer units (the TU format).

```
EX1     TIME
EX2     TIME    TU
```

PROGRAMMING NOTES: The time of day and the date returned to the program will be only as accurate as the corresponding information entered by the operator.

The day of the year is automatically updated at midnight. The operator must reset the day to 1 at the beginning of a new year.

DEC time and the date are returned in formats suitable for unpacking and printing. The date can be unpacked directly. DEC time can be unpacked if the h position (hundredths of seconds) is replaced with a

zone sign. If the hundredths of seconds are important, the time can be unpacked by the insertion of zones between the decimal digits.

STIMER -- Set Interval Timer (R)

The STIMER macro-instruction sets an interval into a programmed interval timer, specifies when the interval timer is to be decremented, and specifies the action to be taken when an interruption signals completion of the interval. The effect of each STIMER macro-instruction issued by a task supersedes the effect of the STIMER macro-instruction issued previously by the same task. In this case, the exit routine that may have been specified by the previous STIMER is not entered.

Name	Operation	Operand
[symbol]	STIMER	{ TASK[,exit-addrx] / REAL[,exit-addrx] / WAIT } , { DINTVL=addrx / BINTVL=addrx / TUINTVL=addrx / TOD=addrx }

TASK
specifies that the interval is to be decremented only when the task issuing the STIMER macro-instruction is in control.

REAL
specifies that the interval is to be decremented continuously, whether or not the task issuing the STIMER is in control.

WAIT
specifies that the interval is to be decremented continuously, and that the task issuing the STIMER is to be placed in a wait condition until an interruption signals the end of the interval.

exit
specifies the address of an exit routine to be given control asynchronously when the specified interval ends. If this operand is omitted when a TASK or REAL interval is specified, the task will be unaware of when the interval has ended. The exit operand should not be written when a WAIT interval is specified; if it is, the supervisor ignores it.

DINTVL
specifies the address of a double-word containing a decimal interval to be set into the timer. The double-word must be aligned on a double-word boundary and contain eight unpacked decimal digits in the format HHMMSSth, where HH = hours in a 24-hour clock, MM = minutes, SS = seconds, t = tenths of seconds, and h = hundredths of seconds. The specified interval must be less than 24 hours.

BINTVL
specifies the address of a full-word containing a binary interval to be set into the timer. The full-word must be aligned on a full-word boundary and contain an unsigned 32-bit binary number in which the least significant bit has a value of 0.01 second. The specified interval must be less than 24 hours.

TUINTVL
specifies the address of a full-word containing a binary interval to be set into the timer. The full-word must be aligned on a

full-word boundary and contain an unsigned 32-bit binary number in which the least significant bit has a value of 1 timer unit. A timer unit is 26 micro-seconds. The specified interval must be less than 24 hours.

TOD

specifies the address of a double-word containing the time of day at which the interval is to end. The double-word must be aligned on a double-word boundary and contain eight unpacked decimal digits in the format HHMMSSth (defined in the DINTVL operand).

This operand is meaningful only when a REAL or WAIT interval is specified. If a TASK interval is specified, the time of day specified will be interpreted as an interval, as though a DINTVL operand were written.

EXCEPTIONAL RETURNS: If WAIT is specified, the waiting task will be dispatched before the end of the interval when necessary to allow execution of an asynchronous exit routine. The routine may be part of the operating system or of the user's problem program, and may be any of the following:

- An input/output or external interruption routine.
- A subtask termination exit routine, specified by the ETXR operand of an ATTACH macro-instruction previously executed by the task.
- A task abnormal exit routine, specified by a STAE macro-instruction previously executed by the task.

Except in the last case (task termination), the task is returned to the wait condition when execution of the exit routine is complete.

ENVIRONMENT: If option 4 was excluded from the system, the entire job step (including any modules that were attached) can have only one active time interval. Each STIMER macro-instruction issued supersedes the effect of any previously issued STIMER macro-instruction. If an STIMER macro-instruction is issued by a timer exit routine, the request is treated as a NOP.

If option 6B was excluded from the system, the STIMER macro-instruction is treated as a NOP.

EXAMPLES: In the following examples, EX1 is used in testing a new loop in a program. The loop should be executed for 6 seconds maximum; therefore, an interval of 6 seconds is specified by the contents (00000600) of the double-word at LOC1. The interval is decremented only when the task is in control. If the interruption occurs, a routine at RTN1 is entered. A TTIMER macro-instruction should be placed after the loop to cancel the interval if execution is successfully completed in less than 6 seconds.

EX2 sets an interval to be decremented continuously, whether or not the task issuing the macro-instruction is in control. The interval is given in the full-word at LOC2. RTN2 is the entry point of the exit routine.

EX3 sets an interval for a program that polls terminals every time the interval expires. The interval is given in the full-word at LOC3. Assuming that the interval is 25 minutes, the task issuing the macro-instruction is placed in a wait condition for 25 minutes; then an interruption occurs and the task again competes for control.

EX4 causes the task to be placed in a wait condition until the time of day specified by the contents of the double-word at LOC4.

```
EX1  STIMER    TASK,RTN1,DINTVL=LOC1
EX2  STIMER    REAL,RTN2,BINTVL=LOC2
EX3  STIMER    WAIT,TUINTVL=LOC3
EX4  STIMER    WAIT,TOD=LOC4
```

PROGRAMMING NOTES: When this macro-instruction is executed, a programmed interval timer is set with the specified interval or with an interval that will provide an interruption at the specified time of day. If TASK is specified, the timer is decremented only when the task issuing the macro-instruction is in control; if REAL or WAIT is specified, the timer is decremented continuously. If TASK or REAL is specified, the task remains in contention for control; if WAIT is specified, the task is placed in a wait condition until after the interruption, and then returned to contention.

If TASK is specified, control is given to the exit routine, if specified, after the interruption. If no exit routine is specified, control returns to the program at the next instruction to be executed, and the program is not notified of the interruption.

If REAL is specified and the task issuing the STIMER macro-instruction is in control when the interruption occurs, control is given to the exit routine or the next instruction to be executed, as for TASK. However, if the task is not in control when the interrupt occurs, the exit routine (if specified), or the next instruction to be executed, is given control when the task regains control normally.

Upon entry to the exit routine specified in an STIMER macro-instruction, register contents are:

Register	Contents
0-1	Internal supervisor information.
2-12	Same as when the interruption occurred.
13	Address of the supervisor-provided save area.
14	Return address (internal supervisor location).
15	Address of the exit routine. (This register can be used to provide addressability.)

Standard linkage conventions, including register saving and restoring responsibilities, apply.

Upon completion of the exit routine, the contents of register 14 must be as they were upon entry to the routine. The exit routine should terminate with a branch to the address in register 14.

One STIMER exit routine can be used by more than one task of a job step. If the routine is used in this manner, it must be reenterable.

Each task can have only one time interval active at a time, but any number of tasks can simultaneously have active time intervals. The time interval specified in the EXEC statement (described in the publication IBM System/360 Operating System: Job Control Language) is an interval set by the system. It cannot be tested or canceled. It does not prevent a task from setting an STIMER interval.

The interval timer cannot accurately measure an interval that is less than the timer resolution. The resolution of the timer -- the time lapse between successive updates -- is the reciprocal of the timer's operating frequency. At a standard line frequency of 60 cps, the resolution is 16.7 milliseconds or 640 timer units; at 50 cps, it is 20 milliseconds or 768 timer units.

TTIMER -- Test Interval Timer (R)

The TTIMER macro-instruction requests the time remaining in the TASK or REAL interval that was previously set by an STIMER macro-instruction issued by the current task. The supervisor places the time in register 0. The TTIMER macro-instruction can also be used to cancel the previously specified interval.

Name	Operation	Operand
[symbol]	TTIMER	[CANCEL]

CANCEL
specifies that the interval in effect should be canceled. If this operand is omitted, processing continues with the unexpired portion of the interval still in effect. If the interval expired before the TTIMER macro-instruction was issued, the CANCEL operand has no effect.

The time remaining in the interval is returned in register 0 whether or not the interval is canceled. It appears as a 32-bit unsigned binary number in which the least significant bit has a value of 1 timer unit. A timer unit is 26 microseconds. The interval is returned in this form even if the interval was originally specified in decimal digits or as a binary multiple of 0.01 second. If the interval expired before the TTIMER macro-instruction was issued, zero is returned in register 0.

ENVIRONMENT: If option 6B was excluded from the system, the TTIMER macro-instruction is treated as a NOP.

EXAMPLES: In the following examples, EX1 requests that the supervisor place in register 0 the amount of time remaining in the interval, and cancel the interval. EX2 requests the amount of time remaining in the interval, and does not cancel the interval.

```
EX1  TTIMER    CANCEL
EX2  TTIMER
```

WTO -- Write to Operator (S)

The WTO macro-instruction writes a message on the operator's console.

Name	Operation	Operand
[symbol]	WTO	message-'text'

message
specifies the message to be written on the console.

The message length must not exceed the line length on the console output device having the shortest line length.

The message appearing on the console does not include the enclosing quotation marks.

EXAMPLES: In the following example, the WTO macro-instruction specifies that the message, FINISHED, is to be written on the operator's console.

```
EX1   WTO   'FINISHED'
```

PROGRAMMING NOTES: The message can include commas, blanks, and quotation marks as in a character constant. The message is assembled into a format-V record, which is the parameter list of the macro-instruction.

L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B except for the following special operand requirements:

Operand	L Form	E Form
message	required	not allowed

WTOR -- Write to Operator with Reply (S)

The WTOR macro-instruction writes a message on the operator's console and enables the operator's reply to be transmitted to the program issuing the macro-instruction.

Name	Operation	Operand
[symbol]	WTOR	message-'text',reply-addr,length-value ,ecb-addr

message
: specifies the message to be written on the console.

  The maximum message length must not exceed the line length on the console output device having the shortest line length.

  The message appearing on the console does not include the enclosing quotation marks.

reply
: specifies the address of an area in main storage into which the message reply text should be placed.

length
: specifies the length, in bytes, of the reply text.

ecb
: specifies the address of an event control block (ECB) representing the completion of the reply.

The program should issue a WAIT macro-instruction to determine when the reply has been transmitted. This WAIT macro-instruction should refer to the event control block whose address is specified by the ecb operand. When the control program has stored the reply in the specified area, it will use the POST macro-instruction to signal completion of the reply.

Refer to the publication IBM System/360 Operating System: Operator's Guide, Form C28-6540, for information about the reply.

CAUTION: When an event control block is posted, its completion flag (bit 1) is set to 1. Before executing a WTOR macro-instruction for a second time, the program must set the completion flag in the event control block to 0. If this is not done, the message reply will appear to have been received before it actually has been.

EXAMPLE: In the following example, the message, A, B, OR C, will be written. The expected reply is one byte long, and will be stored by the control program at the location LOC. ECB1 is the ECB that is to represent completion of the reply. A WAIT macro-instruction referring to ECB1 should be issued to determine when the reply has been received.

```
EX1     WTOR    'A, B, OR C',LOC,1,ECB1
```

PROGRAMMING NOTES: The message can include commas, blanks, and quotation marks as in a character constant. The message is assembled into a format-V record, which is part of the parameter list.

L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B except for the following special operand requirements:

Operand	L Form	E Form
message	required	not allowed

In the E form, a message cannot be specified, but the comma that normally follows the message operand must be written if the reply, length, or ecb operands are written.

WTL -- Write to Log (S)

The WTL macro-instruction writes a message on the system log.

Name	Operation	Operand
[symbol]	WTL	message-'text'

message
specifies the message to be written on the log. The message written on the log does not include the enclosing quotation marks.

ENVIRONMENT: If option 12 was excluded from the system, the WTL macro-instruction is treated as a NOP.

EXAMPLE: In the following example, the message, FINISHED, will be written on the system log.

```
EX1     WTL     'FINISHED'
```

PROGRAMMING NOTES: The message can include commas, blanks, and quotation marks as in a character constant. The message is assembled into a format-V record, which is the parameter list of the macro-instruction.

L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B except for the following special operand requirements:

Operand	L Form	E Form
message	required	not allowed

The macro-instructions contained in this section enable the user to request the data management facilities of the operating system in:

- Creating data sets.
- Gaining access to data.
- Controlling input/output devices.
- Providing buffering.

Since the organization of records for rapid retrieval involves a major decision in data processing, the macro-instructions are grouped to reflect the access methods for the five data organizations: sequential, partitioned, indexed sequential, direct, and telecommunications. An additional grouping contains those macro-instructions that are of general service with all data set organizations. The groupings are as follows:

Data Organization Independent
    General Service Macro-Instructions

Sequential Organization
    Queued Sequential Access Method (QSAM)
    Basic Sequential Access Method (BSAM)

Partitioned Organization
    Basic Partitioned Access Method (BPAM)

Indexed Sequential Organization
    Queued Indexed Sequential Access Method (QISAM)
    Basic Indexed Sequential Access Method (BISAM)

Direct Organization
    Basic Direct Access Method (BDAM)

Telecommunications Organization
    Queued Telecommunications Access Method (QTAM)

A further data capability that can be used with any data organization is provided by the execute channel program feature, which permits the user to program at a level quite close to a device yet within the framework of the operating system. The user must know, and program for, the attributes of the device and the characteristics of the data set residing on it. This capability is represented by the EXCP (execute channel program) macro-instruction, which is discussed in the publication IBM System/360 Operating System: System Programmer's Guide, Form C28-6550.

Direct Access Device Considerations

The System/360 Operating System has defined a standard track format for all direct-access devices. On each track, record zero (Ro) will contain:

Key length	KL=0
Data length	DL=8

The user must take this convention into account when calculating the capacity of a track. Record zero is also referred to as the "capacity record" or "track descriptor record."

ACTUAL ADDRESSING: When the actual address of a block on a direct-access device is returned to the user, it is of the form MBBCCHHR, where

M

specifies the position of an extent entry within the data extent block. This control block is established by the control program during the opening process. M is a one-byte binary number specifying the relative location of the entry within the data extent block. Each extent entry describes a set of contiguous tracks that have been allocated to the data set. Each extent entry contains:

- Start track.
- End track.
- A pointer to a control block that specifies channel and device address.

The extent entries are maintained by the control program and will be established in the sequence that space was originally allocated; i.e., initial allocation followed by each additional allocation in proper sequence.

These extents are created and maintained by the control program and normally do not concern the user. However, if actual addressing is used, M is a required part of the address.

Several access methods offer means by which an actual address may be obtained. In these cases, M is supplied and need only be preserved for subsequent use.

If the user desires to develop actual addresses, he must construct the value for M by performing a series of tests to locate the extent entry in the data extent block that contains the track address he computed.

BBCCHH

specifies cell, cylinder, and head number and collectively form a "track address." Cylinder and head binary designation are recorded on the direct-access device as part of the ID field of each block of data. Refer to the publication IBM 2841 - Control Unit for a description of cell, cylinder, and head number.

R

specifies the block number of a particular block on the designated track. This number is also recorded on the direct-access device as part of the ID field of each block of data.

Cautions: Use of actual addresses will force the user to treat the associated data set as "unmovable."

For sequential processing (QSAM and BSAM), in which the data set resides on more than one volume, only those extent entries for the volume currently being processed are found in the data extent block.

RELATIVE ADDRESSING: There are two types of relative addresses for blocks on direct-access devices. The first type, or relative track address, is of the form TTR, where:

TT
: specifies the relative number of the track on which the block is located. This number is relative to the first track allocated to the data set, for which TT has a value of zero. TT is a two-byte binary number that is unaffected by lack of contiguity for tracks allocated to the data set.

R
: specifies the relative number of the block on track TT. This number is relative to the first block on the track, for which R has a value of zero. R is a one-byte binary number.

The second type of relative address is the relative block address, which specifies the relative number of the block. The number of the block is relative to the first block of the data set, for which the relative block address has a value of zero. The relative block address is a three-byte binary number that is not affected by lack of contiguity for tracks allocated to the data set.

Caution: For sequential processing (BSAM) in which the data set resides on more than one volume, relative addresses can be used only to refer to blocks on the volume currently being processed. A relative track address is relative to the first track allocated to the data set on this volume.

Volume Switching

Volume switching can occur as a result of an end-of-volume condition or the FEOV macro-instruction. In general, the DD statement determines the number of volumes to be associated with a particular data set.

Input: An input operation that detects either a tape mark, an end of data indicator, or an end of last extent causes an end of volume routine to be executed. The user may also issue an FEOV macro-instruction before any of these conditions is detected. In either case, the only factor considered in determining if a volume switch is desired is the number of volume serial numbers made available by the DD statement or the catalog.

Output: Reaching the end of volume on a data set for which output operations are being executed may cause the end of volume routine to be executed, or the user may issue the FEOV macro-instruction. When this occurs, new storage is obtained for the data set on the specific volumes indicated by the DD statement or the catalog. If no volumes are specified (or if more than those specified are required) the new storage is obtained on any available volume (or part of a volume) of the same device type.

End of Data Set Determination: The end-of-data-set exit is taken from the end-of-volume routine for sequential input processing when no further record or block is available. For magnetic tape, note that standard trailer labels have no role in the determination of an end of data set condition.

GENERAL SERVICE MACRO-INSTRUCTIONS

The macro-instructions included in this group provide services that prepare data sets for processing, and main storage for use as buffers and buffer pools. These macro-instructions can be used with all of the access methods presented in this section (except where noted).

Macro-Instruction	Function
DCB	Interfaces with control program
DCBD	Provides symbolic names for data control block
OPEN	Connects the data set to the user's problem program
CLOSE	Disconnects the data set from the user's problem program
FEOV	Forces an end of volume (BSAM and QSAM only)
GETPOOL	Gets a buffer pool
FREEPOOL	Frees a buffer pool
BUILD	Builds a buffer pool
GETBUF	Gets a buffer from a pool
FREEBUF	Returns a buffer to a pool

DCB -- Define a Control Block for Input/Output Operations

The major means of communications between the user and the control program when records are being processed is a data control block. One data control block is required for each data set to be processed concurrently, and contains such information as:

- Characteristics of the data set.
- Types of macro-instructions to be executed.
- Buffering choices.
- Device-dependent options.
- Exit addresses.
- Working storage used by access method routines.

Information required to process a data set is presented to the control program in the DCB macro-instruction. The specific information needed depends upon the data set organization and access method chosen. For this reason, the macro-instruction group for each access method includes a description of a DCB macro-instruction tailored for use with that access method. At the user's discretion, certain information can be supplied by the following alternate sources, in conjunction with the DCB macro-instruction, to complete the data control block:

- The DD statement.
- The data set label.
- The user's problem program -- before opening the data control block (for operands including the DDNAME and EXLST operands), or during the data control block exit routine (for the remaining operands).

Each operand description indicates which of these alternate sources can supply that operand. Until information is supplied by some source, the data control block contains binary zeros, indicating the absence of that parameter.

There are three major considerations common to all variations of the DCB macro-instruction:

- The specification of exits.
- The specification of buffering operands.
- The modification of the data control block by the user to complete the needed information, or to alter the information during execution.

EXITS: Table 6 summarizes the exits that can be specified explicitly by the EODAD or SYNAD operand or implicitly by the EXLST operand in a DCB macro-instruction. The manner in which an exit list is created is shown in Appendix D.

Table 6. Data Management Exits

Type of Exit	When Available	How Specified	Applicable To
Data Control Block	When opening a data control block	EXLST operand and exit list	All access methods
User Label (with standard labels)	When opening or closing data control block or when changing volumes	EXLST operand and exit list	BSAM QSAM
End of Data Set	When no more sequential records/blocks are available	EODAD operand	QSAM BSAM BPAM QISAM QTAM
Error Analysis	After uncorrectable input/output error	SYNAD operand	QSAM BSAM BPAM QISAM QTAM

BUFFERING OPERANDS: Four operands in the DCB macro-instruction are associated with buffering. These operands specify:

- Number of buffers.
- Length of buffers.
- Boundary alignment.
- Address of buffer pool control block.

Some of the buffering operands may not be required when the user specifies other macro-instructions (i.e., BUILD, GETPOOL, GETBUF) that provide buffering services. Whether or not one of these macro-instructions can be issued depends upon the access method being used, since not all access methods support all types of buffering requests. Each DCB macro-instruction contains a summary of the buffering services supported. It also relates the buffering services requested through the general service macro-instructions to the conditions under which the buffering operands are written.

MODIFYING A DATA CONTROL BLOCK: The user can add to or modify the contents of the data control block during the execution of his problem program. These changes may be introduced during one of the following times:

- While the data control block is closed (i.e., before an OPEN macro-instruction is executed or following a CLOSE macro-instruction).
- While the data control block is being opened (i.e., during the data control block exit).
- While the data control block is open.

Note: Only certain fields may be modified while the data control block is open. The appropriate time for changing them is described in each applicable macro-instruction. In general, the EODAD and SYNAD operands must be supplied before the data control block exit routine terminates. However, they can be altered at any time during execution of the user's problem program.

DCBD -- Provide Symbolic Names for a Data Control Block (DCB)

The DCBD macro-instruction generates a DSECT statement that provides a symbolic name for the fields within a data control block. Each field is defined so that, with proper initialization of base registers, the user can refer to any or all fields of one or more data control blocks.

The data control block assembled from a DCB macro-instruction will not have names associated with the individual fields that comprise the control block. To refer or gain access to the fields in the data control block, the user can write a DCBD macro-instruction.

The following conventions have been adopted:

- The name of the dummy control section is IHADCB.
- The name of each field begins with DCB followed by the keyword operand that represents the field in the DCB macro-instruction. If the resulting name is longer than eight characters, it is truncated to eight characters by right-to-left dropout. (The field represented by the operand BLKSIZE would be written DCBBLKSI, for example.)
- The attributes of each data control block field are defined in the dummy control section (DSECT). Note that data control block fields containing addresses are aligned on full-word boundaries. The word contains the address in its three low-order bytes, regardless of the contents of the high-order byte. The length attribute of the symbolic name for each field is four.

Name	Operation	Operand
blank	DCBD	[DSORG=([LR][,PS|BS|QS][,IS][,DA][,PO] [,CX|BX|QX][,MQ])] [,DEVD=([DA][,TA][,PT][,RD|PC][,PR])]

DSORG
specifies the type or types of data control block for which symbolic names are to be defined. The values have the following meanings:

LR - logical-record-length field only (DCBLRECL)
PS - physical sequential organization
BS - physical sequential organization with basic access language
QS - physical sequential organization with queued access language
IS - indexed sequential organization
DA - direct organization
PO - partitioned organization
CX - telecommunications line group
BX - basic telecommunications line group
QX - queued telecommunications line group
MQ - processing program message queue
XE - execute channel program
XA - execute channel program with appendages

The mnemonic PS implies both BS and QS. CX implies both BX and QX. BS and PO define the same type of data control block.

DEVD
specifies a value for the device dependencies that have been coded in the program. The values have the following meanings:

DA - direct-access device
TA - magnetic tape
PT - paper tape
RD - reader/punch or reader
PC - punch
PR - printer

Any combination of these values, expressed as a sublist, is valid with any value of DSORG. If PS, BS, or QS is specified in the DSORG operand and the DEVD operand is omitted, symbolic names will be provided for all possible device dependencies. If LR is specified in the DSORG operand, the DEVD operand is ignored, even if it is specified.

If the operand field of the DCBD macro-instruction is blank, only the symbolic names for the fields of the foundation block will be provided. The foundation block is that section of the data control block that contains the information needed for minimum system support of input/output operations. Before the data control block is opened, the foundation block contains the DCBDDNAM, DCBOFLGS, DCBIFLGS, and DCBMACRF fields. After the data control block is opened, the foundation block contains the DCBTIOT, DCBMACRF, DCBOFLGS, DCBIFLGS, and DCBDEBAD fields. For a description of these fields, refer to the publication IBM System/360 Operating System: System Programmer's Guide, Form C28-6550.

CAUTIONS: The DCBD macro-instruction can be used only once during an assembly. A diagnostic message will be issued if this rule is violated. The macro-instruction may appear at any point in an assembly, or in any number of separate assemblies that are to be combined by the linkage editor. However, if it is written at any location other than at the end of a CSECT or DSECT, the original control section must be resumed by the user. The types of data control blocks requested need not appear in the same assembly as the DCBD macro-instruction.

Because of the many redundant symbols that could result, the operand should not include values that are not required.

EXAMPLE: The following example illustrates how a program can establish the ability to gain access to a field in a data control block. The load address (LA) instruction is used to place the address of the data control block in register 5.

Any use of the symbol definitions provided by IHADCB can be preceded by a USING statement (supplied by the programmer), which establishes a base register for IHADCB. The store operation (STH) will place the half-word value contained in register 6 into the specified field (of the data control block pointed to by register 5). DCBLRECL is the field associated with logical record length.

```
          .
          .
          .
MYDCB     DCB       DDNAME=MYDCB,MACRF=G, (other DCB operands)
          .
          .
          .
          USING     IHADCB,5
          LA        5,MYDCB
          STH       6,DCBLRECL
          .
          .
          .
          DCBD      DSORG=(LR)
```

Note: A DSECT statement is inserted in the assembled program where the DCBD macro-instruction appears. A displacement is established for the DCBLRECL field.

When the DCB macro-instruction is used to provide address fields in a data control block (e.g., the DCBEXLST, DCBEODAD, or DCBSYNAD fields), the location must be contained in the module. However, the problem program can provide address fields by using external references to locations in another module. These external references are resolved when the program is processed by the linkage editor. At that time, the modules are combined to form one load module. An external reference can also be used to provide the address portion of an exit list entry.

Example: An end-of-data set routine named ENDDS is a module that can be used by various programs. To place the address of the routine in the data control block of another module, that module could contain the following:

- A DCBD macro-instruction.
- Appropriate instructions for establishing and loading a base register.
- The instructions:

```
          MVC       DCBEODAD+1(3),VCON
          .
          .
          .
VCON      DC        VL3(ENDDS)
```

PROGRAMMING NOTES: Physical sequential data sets with unlike data attributes can be concatenated for processing by the queued or basic sequential access method. This concatenation must be indicated by setting bit 4 in the DCBOFLGS field of the appropriate data control block. The bit can be set by the instruction OI DCBOFLGS,X'08', provided that a DCBD macro-instruction has been written and the base register specified in the USING statement has been properly loaded.

The OPEN macro-instruction initializes one or more data control blocks so that their associated data sets can be processed.

Name	Operation	Operand
[symbol]	OPEN	({dcb-addr,[(opt_1-code[,opt_2-code])],}...)

dcb
specifies the address of the data control block to be initialized.

opt_1
specifies the intended method of input/output processing for the associated data set. The values have the following meanings:

Code	Meaning
INPUT	input data set. This value is assumed if opt_1 is omitted.
OUTPUT	output data set.
INOUT	input data set first, and, without reopening, output data set (BSAM only).
OUTIN	output data set first, and, without reopening, input data set (BSAM only).
RDBACK	positions an input data set to be read backward (BSAM and QSAM, magnetic tape only).
UPDAT	allows updating of the data set in place (direct-access devices only).

The introduction to each access method lists the relationships between the value selected for opt_1 and the actions of the macro-instructions.

If the DD statement disposition subparameter is MOD, opt_1 must be OUTPUT or OUTIN for the MOD to be effective.

opt_2
specifies the volume disposition that is to be provided when volume switching occurs. The values have the following meanings:

Code	Meaning
DISP	tests the disposition given in the DD control statement and provides appropriate positioning. Refer to the publication IBM Operating System/360: Job Control Language for a description of the DD statement. This value is assumed if the opt_2 operand is omitted.
REREAD	repositions the volume to process the data set again (QSAM and BSAM).

LEAVE performs no additional positioning at end-of-volume processing (QSAM and BSAM).

The opt_2 operand may be specified only if opt_1 is also specified. The opt_2 operand is applicable to the volume positioning of magnetic tape and direct-access devices. It will be ignored if other devices are used. If the number of volumes exceeds the number of available units, opt_2 will be ignored.

EXAMPLES: In the following examples, EX1 results in the data control block INVEN being opened for an input data set. EX2 results in the two data control blocks INVEN and REPORT being opened with different options. EX3 results in the two data control blocks INVEN and MASTER being opened; they are opened for input data sets since INPUT is assumed when opt_1 is omitted.

```
EX1     OPEN    (INVEN,(INPUT))
EX2     OPEN    (INVEN,(INPUT),REPORT,(OUTPUT,LEAVE))
EX3     OPEN    (INVEN,,MASTER)
```

PROGRAMMING NOTES: Any number of data control block addresses and associated options may be specified in the OPEN macro-instruction. This facility allows parallel opening of the data control blocks and their associated data sets. One of the services performed at this point is the processing of labels. Appendix E describes standard secondary storage label formats.

If a data-control-block exit routine or a user-label exit routine is to be executed, the exit list (DCBEXLST) address must be provided in the appropriate data control block.

The format of the exit list, use of the exit list during the opening process, and exit routine requirements are discussed in Appendix D.

The user may allow the control program to obtain a buffer pool for a data control block during the opening process. This option is described in the DCB macro-instruction for each access method.

The parameter list resulting from expansion of the OPEN macro-instruction contains a full-word entry for each data control block and its associated options. The three low-order bytes of each word contain the 24-bit address of a data control block. The high-order byte contains a code, as follows:

Bit	Binary Contents	Meaning
0	0	Another parameter follows
	1	Last entry in list
1	--	(Reserved)
2-3	00	Use DD control statement disposition
	01	Position volume for REREAD
	11	Position volume for LEAVE
4-7	0000	INPUT
	1111	OUTPUT
	0011	INOUT
	0111	OUTIN
	0001	RDBACK
	0100	UPDAT

CAUTIONS: The following errors will cause the results indicated:

Error	Result
Opening a data control block that is already open.	No action
Attempting to open a data control block when the dcb operand does not specify the address of a data control block.	Unpredictable
Opening a data control block when a corresponding DD statement has not been provided.	No action; however, an attempt to use the data set results in abnormal termination of the task

The last of these errors can be detected by testing bit 3 of the OFLGS field in the data control block. Bit 3 is set to 0 in the case of an error, and can be tested by the sequence:

```
TM  DCBOFLGS,X'10'
BZ  ERRORRTN          (Branch to user's error routine)
```

provided that a DCBD macro-instruction has been written and the base register specified in the USING statement has been properly loaded.

L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B.

CLOSE -- Disconnect Data Set from User's Problem Program (S)

The CLOSE macro-instruction disconnects one or more data sets from the user's problem program.

Name	Operation	Operand
[symbol]	CLOSE	({dcb-addr,[opt-code],}...)

dcb
: specifies the address of the data control block opened for the data set whose processing is to terminate.

opt
: specifies the volume disposition that is to occur as a result of closing. Its values and meanings are as follows:

Code	Meaning
DISP	tests the disposition given in the DD control statement and provides the appropriate positioning. Refer to the publication IBM System/360 Operating System: Job Control Language for a description of the DD statement. This value is assumed if the opt operand is omitted.

REREAD positions the current volume to process the data set again.

LEAVE positions the current volume to the logical end of the data set just processed.

The opt operand is applicable to the volume disposition of magnetic tape or direct-access devices only; it will be ignored if other devices are used.

CAUTIONS: The following errors will cause the results indicated:

Error	Result
Closing a data control block that is already closed.	No action
Closing when the dcb operand does not specify the address of a data control block.	Unpredictable

EXAMPLES: In the following examples, EX1 results in the data set associated with the data control block INVEN being closed with no repositioning. EX2 results in the two data sets associated with the data control blocks INVEN and REPORT being closed with different options. EX3 results in data sets associated with two data control blocks being closed. Since opt is omitted in EX3, the volume dispositions indicated on the DD statements are effective.

```
EX1   CLOSE   (INVEN,LEAVE)
EX2   CLOSE   (INVEN,LEAVE,REPORT,REREAD)
EX3   CLOSE   (INVEN,,MASTER)
```

PROGRAMMING NOTES: Any number of data control block addresses and associated options may be specified in the CLOSE macro-instruction. This facility makes it possible to close data control blocks and their associated data sets in parallel.

For magnetic tape, positioning will vary, depending on whether or not the data set uses labels. Table 7 defines a position number for labeled and unlabeled tapes and Table 8 relates the options chosen in the OPEN and CLOSE macro-instructions to the positioning of tape volumes.

Table 7. Magnetic Tape Positions - QSAM and BSAM

Position	Labeled Tape	Unlabeled Tape
1	Preceding data set header label group	Preceding first data block of the data set
2	Following tape mark that terminates trailer label group of data set	Following tape mark that terminates last data block of data set

Table 8. Factors Determining Magnetic Tape Positioning - QSAM and BSAM

Apply to		Opt_1 of OPEN Specified as	Other Factors Influencing Positioning	Direction of Last Input Operation	Positioning as Specified by Opt in CLOSE	
QSAM	BSAM				LEAVE	REREAD
X	X	OUTPUT		Not applicable		
	X	OUTIN		Not determining factor		
	X	INOUT	At least one WRITE operation	Not determining factor	Position 2	Position 1
X	X	INPUT		Forward		
	X	INOUT	No WRITE operation executed	Forward		
	X	RDBACK		Forward		
X	X	INPUT		Backward		
	X	INOUT	No WRITE operation executed	Backward	Position 1	Position 2
X	X	RDBACK		Backward		

The parameter list resulting from expansion of the CLOSE macro-instruction contains a full-word entry for each data control block with its associated options. The three low-order bytes of each word contain the 24-bit address of a data control block. The high-order byte contains a code, as follows:

Bit	Binary Contents	Meaning
0	0	Another parameter follows
	1	Last entry in list
1	--	(Reserved)
2-3	00	Use DD control statement disposition
	01	Position volume for REREAD
	11	Position volume for LEAVE
4-7	--	(Ignored)

L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B. The E form of the CLOSE macro-instruction can refer to a list generated by the L form of the OPEN macro-instruction.

FEOV -- Force End of Volume (R)

The FEOV macro-instruction directs the control program to advance to the next volume of a data set before the physical end of the current volume is reached. This macro-instruction is applicable only to data sets processed by the queued and basic sequential access methods (magnetic tape and direct-access devices only).

Name	Operation	Operand
[symbol]	FEOV	{dcb-addrx / (1)}

dcb
specifies the address of the data control block that is opened for the data set.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTIONS: The following errors will cause the results indicated:

Error	Result
Forcing an end of volume when the dcb operand specifies the address of a data control block that is not open	No action
Forcing an end of volume when the dcb operand does not specify the address of a data control block	Unpredictable

When BSAM is used, all read or write operations must be checked for completion before the FEOV macro-instruction is executed.

EXAMPLE: In the following example, the control program is directed to advance to the next volume of the data set associated with the data control block REPORT.

```
EX1  FEOV      REPORT
```

GETPOOL -- Get a Buffer Pool (R)

The GETPOOL macro-instruction requests allocation of an area of main storage, and constructs a buffer pool in the allocated area. The buffer pool is assigned to the specified data control block.

Name	Operation	Operand
[symbol]	GETPOOL	{dcb-addrx / (1)},{number-value,length-value / (0)}

dcb
specifies the address of the data control block to which the buffer pool is to be assigned.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

number
specifies the number of buffers to be in the pool. The maximum value is 255.

length
specifies the number of bytes in each buffer. The value will be increased, if necessary, to a multiple of double-words by the control program. The maximum value is 32,760.

If (0) is written, the value giving the number of buffers must have been loaded into the two high-order bytes of parameter register 0, and the value specifying the length of each buffer into the two low-order bytes, before execution of this macro-instruction.

CAUTIONS: The following cautions apply:

1. Only one buffer pool may be assigned to a data control block.

2. A GETPOOL macro-instruction may be issued either before the data control block is opened or during execution of the data control block exit routine during the opening process.

3. In response to the GETPOOL macro-instruction, a data management routine will issue the equivalent of a GETMAIN macro-instruction. The user must be familiar with the cautions relevant to the GETMAIN macro-instruction. (Refer to Section 2.)

EXAMPLES: In the following examples, EX1 constructs two buffers, of 136 bytes each, in an allocated area of main storage. This buffer pool is assigned to the data control block REPORT. EX2 indicates that the required parameters were loaded into registers 1 and 0 before execution of the macro-instruction.

```
EX1   GETPOOL   REPORT,2,136
EX2   GETPOOL   (1),(0)
```

PROGRAMMING NOTES: A buffer pool consists of one eight-byte buffer pool control block followed by the specified number of buffers. Each buffer is aligned on the boundary type specified in the data control block field DCBBFALN.

The FREEPOOL macro-instruction should be issued to return the allocated main storage to the system.

FREEPOOL -- Free a Buffer Pool (R)

The FREEPOOL macro-instruction releases an area of main storage that had previously been assigned as a buffer pool for a specified data control block. The area must have been acquired by the execution of a GETPOOL macro-instruction or automatically by the control program when the data control block was opened.

Name	Operation	Operand
[symbol]	FREEPOOL	{dcb-addrx / (1)}

dcb
specifies the address of the data control block to which the buffer pool was assigned.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTIONS: The following cautions apply:

1. If the associated data control block is being processed by means of QSAM with simple buffering, FREEPOOL should not be issued until the data control block is closed. If exchange buffering is used with QSAM, FREEPOOL should not be issued until all the data control blocks that exchanged buffer segments with the specified buffer pool are also closed.

2. If the associated data set is processed by means of BSAM, FREEPOOL may be issued as soon as the buffers are no longer required.

3. If the associated data set is processed by means of QISAM, FREEPOOL should not be issued until all the data control blocks are closed.

4. In all other cases, FREEPOOL should be issued at the earliest possible time to permit the storage area to be released.

EXAMPLES: In the following examples, EX1 returns the buffer pool assigned to the data control block OUTPUT to the system's available main storage. EX2 returns the buffer pool assigned to the data control block whose address is in register 1.

```
EX1   FREEPOOL   OUTPUT
EX2   FREEPOOL   (1)
```

BUILD -- Build a Buffer Pool (R)

The BUILD macro-instruction constructs a buffer pool in an area of main storage provided by the user.

Name	Operation	Operand
[symbol]	BUILD	{pool-addrx / (1)},{number-value,length-value / (0)}

pool
specifies the address of an area of main storage to be used as a buffer pool. This area must be aligned on a full-word boundary.

If (1) is written, the pool address must have been loaded into parameter register 1 before execution of this macro-instruction.

number
specifies the number of buffers to be in the pool. The maximum value is 255.

length
specifies the number of bytes in each buffer. The value will be rounded to the next highest multiple of full-words by the control program. The maximum value is 32,764.

If (0) is written, the value specifying the number of buffers must have been loaded in the two high-order bytes of parameter register 0, and the value specifying the length of each buffer into the two low-order bytes, before execution of this macro-instruction.

CAUTIONS: The area of main storage provided by the user must be large enough to contain both an eight-byte buffer pool control block and the specified number of buffers after the length of each buffer has been rounded to the next highest multiple of full words. If the buffer pool is to be assigned to a data control block whose data set requires double-word alignment for each buffer, the area provided by the user should be aligned on a double-word boundary, and the length of each buffer should be a double-word multiple.

EXAMPLES: In the following examples, EX1 constructs a buffer pool containing five buffers, of 100 bytes each, in the area beginning at POOLLOC. EX2 indicates that the required parameters were loaded into registers 1 and 0 before execution of the macro-instruction.

```
EX1   BUILD   POOLLOC,5,100
EX2   BUILD   (1),(0)
```

GETBUF -- Get a Buffer From a Pool (R)

The GETBUF macro-instruction obtains a buffer from a buffer pool.

Name	Operation	Operand
[symbol]	GETBUF	{dcb-addrx / (1)},register-absexp

dcb
specifies the address of the data control block opened for the data set being processed.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

register
specifies a general register (2-12) into which the control program is to place the address of the buffer.

CAUTIONS: A buffer pool must have been previously assigned to the data control block by the use of the BUILD, GETPOOL or OPEN macro-instructions.

Buffers must be returned to the pool by the FREEBUF macro-instruction.

EXCEPTIONAL RETURNS: A return is always made to the user's next instruction. If no buffer is available within the pool, the register specified by the user will contain zero rather than an address.

EXAMPLE: In the following example, the BUILD macro-instruction is used to structure the 8008-byte area IOPOOL into 10 buffers of 800 bytes each (preceded by an eight-byte buffer pool control block). The GETBUF macro-instruction is used to obtain the address of an available buffer in register 5. That buffer is then used to hold an input block when a format-F data set is being read. (The length operand is not required in the READ macro-instruction.)

```
INDCB    DCB      BUFCB=IOPOOL (and other operands)
          .
          .
          .
         BUILD    IOPOOL,10,800
          .
          .
          .
         GETBUF   INDCB,5
          .
          .
          .
         READ     DECB1,SF,INDCB,(5)
          .
          .
          .
         DS       0D
IOPOOL   DS       2002F
          .
          .
          .
```

FREEBUF -- Return a Buffer to a Pool (R)

The FREEBUF macro-instruction is used to return a buffer to a pool maintained for the GETBUF macro-instruction.

Name	Operation	Operand
[symbol]	FREEBUF	{dcb-addrx / (1)},register-absexp

dcb
specifies the address of the data control block opened for the data set.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

register
specifies the general register (2-12) that contains the address of the buffer being returned to the pool.

CAUTIONS: A buffer pool must have been assigned to the data control block, and the specified buffer must have been obtained by a GETBUF macro-instruction. The action of the macro-instruction is unpredictable in other cases.

QUEUED SEQUENTIAL ACCESS METHOD (QSAM)

The queued sequential access method (QSAM) is, for the most part, device-independent, permitting programs to be written to use one of a number of different input/output devices. Records of a sequential data set can be stored and retrieved without the writing of blocking/deblocking and buffering routines by the user. Either simple or exchange buffering can be requested.

QSAM does not process keys on direct-access devices. Any request to process a data set with keys will result in gaining access to only the data portion of each block.

The QSAM device-independent macro-instructions (GET, RELSE, PUT, PUTX, TRUNC, and PRTOV), device-dependent macro-instruction (CNTRL), and general service macro-instructions (BUILD, DCB, GETPOOL, FREEPOOL, OPEN, CLOSE, and FEOV) are used with the queued sequential access method.

QSAM responds to control characters when logical records are written to a printer or card punch. These control characters are listed in Appendix F, which also contains a discussion of SYSOUT writers.

Macro-Instruction	Function
DCB	Constructs a data control block for a sequential data set.
GET	Gets a logical record from a sequential data set.
PUT	Includes a logical record in an output data set.
PUTX	Returns an updated record to a sequential data set or includes a record of an input data set in an output data set.
RELSE	Causes the remaining logical records in a buffer to be ignored.
TRUNC	Causes the next logical record to be written as the first record of the next block.
CNTRL	Controls a printer or card reader.
PRTOV	Tests for printer carriage overflow.

The OPEN macro-instruction option specifying the intended method of input/output processing, opt_1, has the following effect on the QSAM macro-instructions:

OPEN Macro-Instruction Operands	Effect
INPUT	A PUT or PUTX may not be used.
OUTPUT	A PUT or PUTX will destroy the entire data set that follows the record written.
RDBACK (magnetic tape devices only)	A PUT or PUTX may not be used.
UPDAT (direct-access devices only)	A PUTX will replace the record last retrieved by a GET, or a record presented by the user as a replacement.

Table 9. Buffering and Modes of GET-PUT

INPUT DATA SET GET \ OUTPUT DATA SET PUT	SIMPLE BUFFERING locate-mode	SIMPLE BUFFERING move-mode	EXCHANGE BUFFERING substitute-mode	EXCHANGE BUFFERING move-mode
SIMPLE BUFFERING locate-mode	•User must move data	•PUTX output mode can be used •Control program moves data to output buffer	•Work area required •User must move data to work area	•Control program moves data to output buffer •PUTX output mode can be used
SIMPLE BUFFERING move-mode	•Control program moves data •No other data movement for sequence: PUT locate-mode, GET move-mode	•Work area required •Control program moves data twice	•Work area required •Control program moves data into work area	•Work area required •Control program moves data twice
EXCHANGE BUFFERING locate-mode	•User must move data	•PUTX output mode can be used •Control program moves data to output buffer	•Work area required •User must move data into work area	•PUTX output mode can be used-No data movement •Control program moves data for PUT move-mode
EXCHANGE BUFFERING substitute-mode	•Work area required •User must move data to output buffer	•Work area required •Control program moves data to output buffer	•Work area required •No movement of data	•Work area required •Control program moves data to output buffer

Table 9 summarizes the valid combinations of buffering methods and modes of operation when GET and PUT macro-instructions are used to copy a data set (possibly with insertions and deletions). Two or more data control blocks may be involved, as in the case of an operation that merges input data sets into a single output data set.

Once the user has selected the input and output buffering methods and modes of operation, Table 9 also indicates the requirement for data movement and any special requirements. For example, if the output data set uses exchange buffering and the substitute mode of operation, all data movement can be avoided by using exchange buffering with the substitute mode of operation for the input data set.

DCB - Define Data Control Block for QSAM

The DCB macro-instruction reserves space for a data control block and informs the control program of the characteristics and intended uses of a data set.

Name	Operation	Operand
[symbol]	DCB	DSORG={PS|PSU},MACRF=code [,DDNAME=symbol][,DEVD=code] [,OPTCD={W|C|WC}][,RECFM=code] [,LRECL=absexp][,BLKSIZE=absexp] [,BFTEK={S|E}][,BUFNO=absexp] [,BFALN={F|D}][,BUFL=absexp] [,BUFCB=relexp][,EODAD=relexp][,EXLST=relexp] [,SYNAD=relexp][,EROPT={ACC|SKP|ABE}]

The keyword operands DSORG and MACRF can be supplied by only the DCB macro-instruction. The remaining operands can be supplied after assembly time by other sources; these sources are indicated in the operand descriptions.

DSORG
specifies the organization of the data set as one of the following:

PS - a physical sequential organization

PSU - a physical sequential organization in which any data set contains location-dependent information with respect to this data set. The data set is unmovable.

MACRF
specifies the types of macro-instructions that will be used in processing the data set, as follows:

```
                  (G{M|L|T|MC|LC|TC})
,MACRF={ (P{M|L|T|MC|LC|TC})                    }
        (G{M|L|T|MC|LC|TC},P{M|L|T|MC|LC|TC})
```

G - GET macro-instruction (and implies RELSE macro-instruction)
P - PUT or PUTX macro-instruction (and implies TRUNC macro-instruction)
M - move-mode operation
L - locate-mode operation
T - substitute-mode operation
C - CNTRL macro-instruction

Note: Only the move mode can be used with data sets on paper tape.

DDNAME

specifies the name of the DD statement that will be used to describe the data set to be processed.

This information can also be supplied by the user's problem program before opening the data control block.

DEVD

specifies the device or devices on which the data sets may reside. For certain devices, additional keyword operands can be written to provide device-dependent information. The format of these operands is as follows:

```
                 DA
                 TA[,DEN={0|1|2}][,TRTCH={C|E|T|ET}]
,DEVD=           PT[,CODE={I|F|B|C|A|T|N}]
                 PR[,PRTSP={0|1|2|3}]
                 {PC}[,MODE={C|E}][,STACK={1|2}]
                 {RD}
```

For each device type, a device-dependent area is reserved in the data control block. The device types, from DA to RD, are listed in the above format in order of descending space requirements. To achieve program device independence over a limited set of devices up to the time of execution, the programmer should specify the device type requiring the largest area. If this operand is omitted, the maximum device-dependent area is reserved.

Each of the additional keyword operands is described under the optional value of the DEVD operand with which it belongs. Note that the alternate sources of information for these operands do not apply to the DEVD operand.

DA: specifies a direct-access device.

TA: specifies magnetic tape.

DEN

can be used with magnetic tape, and specifies a value for the tape recording density in bits per inch as listed in Table 10.

This information can be supplied by the DD statement or the user's problem program. If not supplied by any source, the lowest density is assumed.

Table 10. DEN Values

DEN Value	Tape Recording Density (bits/inch)		
	Model 2400		Model 7340
	7 Track	9 Track	
0	200	-	1511
1	556	-	3022
2	800	800	-

TRTCH
is used with seven-track tape to specify the tape recording technique, as follows:

C - specifies that the data conversion feature is to be used; if data conversion is not available, only format-F and -U records are supported by the control program.
E - specifies that even parity is to be used; if omitted, odd parity is assumed.
T - specifies that BCDIC to EBCDIC translation is required.

This information can be supplied by the DD statement or the user's problem program.

PT: specifies paper tape.

CODE
can be used with paper tape, and specifies the code in which the data was punched as follows:

I - IBM BCD perforated tape and transmission code (8 tracks)
F - Friden (8 tracks)
B - Burroughs (7 tracks)
C - National Cash Register (8 tracks)
A - ASCII (8 tracks)
T - Teletype (5 tracks)
N - no conversion (format-F records only)

This information can be supplied by the DD statement or the user's problem program. If not supplied by any source, I is assumed.

The following apply when conversion is requested:

- Characters that are deleted in the conversion process are not counted in determining the block size.
- A character determined to have a parity error will not be converted when the record is moved to the user's input area.

PR: specifies printer.

PRTSP
specifies the line spacing on a printer as 0, 1, 2, or 3. (This operand is valid if control characters are not specified in the DCBRECFM field of the data control block.)

This information can be supplied by the DD statement or the user's problem program. If not supplied by any source, 1 is assumed.

PC: specifies a card punch.

RD: specifies a card reader or card read punch.

MODE
can be used with a card reader, a card punch, or a card read punch and specifies the mode of operation as follows:

C - the card image (column binary) mode
E - the EBCDIC code

This information can be supplied by the DD statement or the user's problem program. If not supplied by any source, E is assumed.

STACK
can be used with a card reader, a card punch, or a read punch and specifies which stacker bin is to receive the card. Either 1 or 2 is specified.

This information can be supplied by the DD statement or the problem program. If not supplied by any source, 1 is assumed.

OPTCD
specifies an optional service to be provided by the control program, as follows:

W - perform a write validity check (on direct-access devices only).
C - process using the chained scheduling method.
WC - perform a validity check and use chained scheduling.

This information can be supplied by the DD statement or the user's problem program. If not supplied by any source, none of the services are provided.

RECFM
specifies the characteristics of the records in the data set, as follows:

$$,\text{RECFM}=\begin{cases}\text{U[T][A|M]}\\ \text{V[B|T][A|M]}\\ \text{F[B|S|T|BS|BT|BST|ST][A|M]}\end{cases}$$

where the record format is:

U - undefined records
V - variable-length records
F - fixed-length records

the physical attributes are:

B - blocked records
S - standard blocks; no truncated blocks or unfilled tracks within the data set, with the possible exception of the last block or track
T - track overflow is to be used

and the record contains:

A - ASA control character (Refer to Appendix F.)
M - machine code control character (Refer to Appendix F.)

For paper tape records, format-U records imply the use of end-of-record (EOR) characters to delimit the block. The only valid formats on paper tape are U and unblocked F.

Record format information (F, V, and U) can be supplied by any of the three possible alternate sources. The absence of any of the physical attribute mnemonics (B, S, and T) implies the opposite of that attribute. If no record format information is supplied, a format-U record without a control character is assumed.

LRECL
specifies the length, in bytes, of a format-F logical record or the maximum length of a format-V logical record. This operand is omitted for format-U records but must be supplied for format-F and -V records. The maximum value is 32,760.

This information can be supplied by any of the alternate sources.

When reading format-U records, the record length is placed in this field by the control program after each GET macro-instruction. The record length is also provided in this manner when reading format-F records with code conversion from paper tape.

BLKSIZE
specifies the maximum length, in bytes, of a block. For format-F records, the length must be an integral multiple of the LRECL value. For format-V records, the length must include the four-byte block-length field that is recorded at the beginning of each block. The maximum value is 32,760.

This information can be supplied by any of the three alternate sources.

When reading format-U records from paper tape, the number of characters moved to the user's input area is limited either by the number specified in the DCBBLKSI field or by the occurrence of an EOR character, whichever comes first. This could result in a movement of zero characters, indicating no data for that particular request. When reading format-F records from paper tape, the physical end of the tape must coincide with the end of a block, or a wrong length indication is given. For format-U records, the physical end of the tape is treated as an EOR character.

NOTE: Refer to Table 11 for a list of the situations in which the following five operands (BFTEK, BUFNO, BFALN, BUFL, and BUFCB) are applicable.

BFTEK
specifies the type of buffering to be supplied by the control program, as follows:

S - simple buffering
E - exchange buffering

This information can be supplied by the DD statement or the user's problem program. BFTEK and BFALN information must be supplied by the same source.

Note: Exchange buffering cannot be used with format-V blocked records.

Table 11. QSAM Buffer Acquisition and Data Control Block Field Requirements

Usage and Characteristics		Method of Obtaining Buffer Pool		
		Automatically By OPEN	BUILD	GETPOOL
When Issued			In data control block exit routine or before OPEN	In data control block exit routine or before OPEN
Result		System acquires storage and structures into buffer pool	Structures storage into buffer pool	Acquires storage and structures into buffer pool
Data Control Block Field Requirements (to be provided no later than conclusion of data control block exit routine)	DCBBUFNO	Optional[1]	Required; user sets this field before or after BUILD is executed	Ignored; GETPOOL sets this field
	DCBBUFCB	Must be omitted; OPEN sets this field	Required; user sets this field before or after BUILD is executed	Ignored; GETPOOL sets this field
	DCBBFALN	Optional[2]	Ignored	Optional[2]
	DCBBUFL	Optional; if omitted, the field is not altered and DCBBLKSI is used	Ignored	Ignored
	DCBBFTEK	Required	Required	Required
Features		Provides standard options	More than one data control block can use pool	Execution time request for storage
Cautions		Only one data control block can use pool; must use FREEPOOL	User responsible for storage acquisition and boundary alignment; must close all data control blocks before reissuing BUILD	Only one data control block can use pool; must use FREEPOOL

[1]If omitted, the field is not altered and three buffers are provided for 2540 Card Read Punch; two for all other devices.
[2]If omitted, the field is not altered and double-word alignment is assumed.

BUFNO
specifies the number of buffers to be assigned to the data control block. The maximum number that can be specified is 255; however,

the number must not exceed the limit on input/output requests established during system generation.

This information can be supplied by the DD statement or the user's problem program.

BFALN
specifies the boundary alignment, in bytes, of each buffer, as follows:

F - the buffer starts on a full-word boundary (one that is not also a double-word boundary)
D - the buffer starts on a double-word boundary.

This information can be supplied by the DD statement or the user's problem program. BFALN and BFTEK information must be supplied by the same source.

BUFL
specifies the length in bytes of each buffer to be obtained for a buffer pool. The maximum value is 32,760.

This information can be supplied by the DD statement or the user's problem program. If it is not supplied, the control program calculates the length by using the value supplied for the BLKSIZE operand.

BUFCB
specifies the address of a buffer pool control block (i.e., the eight-byte field preceding the buffers in a buffer pool).

This information can be supplied by the user's problem program.

EODAD
specifies the address of the user's end-of-data set exit routine for input data sets. This routine is entered when the user requests a record and there are no more records to be retrieved. If no routine has been provided, the task is abnormally terminated.

The only alternate source for this information is the user's problem program.

EXLST
specifies the address of an exit list created by the programmer. The format of the list is shown in Appendix D.

Exit lists are required if:

- User label exit routines or data control block exit routines are used.
- A checkpoint is to be taken automatically at the beginning of each volume (except the first).

The alternate source for this information is the user's problem program.

SYNAD
specifies the address of the user's synchronous error exit routine. The routine is entered if input/output errors result from an attempt to process data records. If no routine is specified and an error occurs, the option specified by the EROPT parameter is executed.

The only alternate source for this information is the user's problem program.

EROPT
specifies the option to be executed if an error occurs and either there is no synchronous exceptional error (SYNAD) exit routine or there is a SYNAD routine and the programmer wishes to return from it to his processing program. One of the following is specified:

ACC - accept error block
SKP - skip error block
ABE - terminate the task

Table 12 indicates the choices that are permitted for each type of data set processing.

This information can be supplied by the DD statement or the user's problem program. If not supplied by any source, ABE is assumed.

Table 12. Error Options for QSAM

Operand	Process Data Set for		
	INPUT,RDBACK	OUTPUT	UPDATE
ACC	X	X[1]	X
SKP	X		X
ABE	X	X	X

[1] Valid for printer only.

CAUTION: The DCB macro-instruction must not be coded within the first 16 bytes of a control section. It can be preceded by padding, constants, or instructions.

GET -- Locate Mode (R)

This GET macro-instruction locates the next sequential logical record to be processed. The user can process the record within the input buffer or move the record to a work area.

Name	Operation	Operand
[symbol]	GET	{dcb-addrx / (1)}

dcb
specifies the address of the data control block opened for the data set being processed.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

The control program returns the address of the next record in parameter register 1, and places the record length in the logical record

length (DCBLRECL) field of the data control block. When chained scheduling is used with format-U records, the maximum record length is placed in the DCBLRECL field.

EXCEPTIONAL RETURNS: The end-of-data set routine specified in the EODAD field of the data control block is given control when the user issues a GET macro-instruction after all the records in the data set have been processed. The user can only close the data set. (Unpredictable results will occur if the user issues a GET or PUT macro-instruction).

After a GET macro-instruction is issued, the synchronous error exit (SYNAD) routine specified in the data control block is given control if either of the following conditions exists:

- The next record to be processed starts a block that could not be read satisfactorily because of an error condition.
- A preceding PUT or PUTX macro-instruction could not be executed without resulting in an error condition. This situation is discovered by the GET macro-instruction.

When the SYNAD routine is given control, the general registers will be set as shown in Table 13.

Table 13. Register Contents Upon Entry to SYNAD Routine

Register	Bit	Contents
0	8-31	Address of a location containing standard status information.
	0-7	A displacement value that can be added to the above address to provide the address of the first full-word of the channel command word (CCW) that points to the current buffer. This full-word contains the address of the buffer in bits 8 through 31. If exchange buffering with data chaining is used, the CCW is the first of a list in which each CCW points to a single buffer segment. The chained data flag of all but the last CCW in the list will be set.
1	0	Set to 1 if error was caused by GET.
	1	Set to 1 if error was caused by PUT.
	2	Not used.
	3	Set to 1 if (1) error indicated by bit 0 did not prevent reading of the block, or (2) error indicated by bit 1 occurred during update of an existing block. Set to 0 if error prevented reading of block or occurred during creation of a new block.
	4	Not used.
	5	Set to 1 if undefined characters encountered in translation from paper tape.
	6-7	Not used.
	8-31	Address of the data control block.
2-13		The contents that existed before the macro-instruction was executed.
14		The return address.
15		The address of the SYNAD routine.

Bits 8 through 31 of register 0 contain the address of a block of standard status information that can be interrogated in the SYNAD exit routine. This information includes the channel status word that describes the detected error. The information of interest to the programmer begins two bytes from the address in the register. Refer to Appendix G for the standard status information.

The programmer can return from the SYNAD routine using the RETURN macro-instruction. The control program will then execute the option specified in the EROPT field of the data control block. If the user does not return, the data set can only be closed.

If the user has not specified a synchronous error exit routine and an error condition is discovered, the control program will execute the option specified in the EROPT field. Note that if an error block is not in main storage (bit 3 set to 0 and bit 0 set to 1), the action of the ACC option is unpredictable.

EXAMPLE: In the following example, the address of the data control block INDCB is loaded into parameter register 1 before the GET macro-instruction is executed. Special register notation, (1), is written in the operand field of the macro-instruction rather than an address, to reflect the manner in which information is to be passed to the control program. After the GET macro-instruction has been executed, register 1 contains the address of the next record.

```
         .
         .
         .
LA 1,INDCB
         .
         .
         .
GET (1)
         .
         .
         .
```

GET -- Move Mode (R)

This GET macro-instruction moves the next sequential logical record to the user's work area.

Name	Operation	Operand
[symbol]	GET	{dcb-addrx | (1)}, {area-addrx | (0)}

dcb
specifies the address of the data control block opened for the data set being processed.

If (1) is written, the address must have been loaded into register 1 before execution of this macro-instruction.

area
specifies the address of the user's work area to which the control program will move logical records.

If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

The control program returns the address of the work area containing the logical record in register 1. (This feature provides compatibility with the substitute-mode GET macro-instruction, and allows move mode to be used by the control program when substitute mode cannot be supported.) The control program also returns the record length in the logical record length (DCBLRECL) field of the data control block. When chained scheduling is used with format-U records, the maximum record length is placed in the DCBLRECL field.

CAUTIONS: The move-mode GET macro-instruction can be used with all record formats, but only with simple buffering. (Refer to Table 9.)

When chained scheduling is used with format-U records, the work area must be large enough to contain a maximum-length record; the number of bytes specified by DCBBLKSI is moved from the input buffer to the work area.

EXCEPTIONAL RETURNS: Refer to the locate-mode GET macro-instruction.

EXAMPLE: in the following example, the next record from the data set associated with the data control block STAT is moved to the work area SAMPLES. The address of the work area is returned to the user in parameter register 1.

```
EX1    GET     STAT,SAMPLES
```

GET -- Substitute Mode (R)

This GET macro-instruction transfers ownership of the next sequential record in a data set from the control program to the user. In return, the ownership of a work area is transferred from the user to the control program for future use as an input buffer. There is no movement of data. The work area should be the same size as the buffer segment containing the record.

Name	Operation	Operand
[symbol]	GET	{dcb-addrx | (1)}, {area-addrx | (0)}

dcb
: specifies the address of the data control block opened for the data set being processed.

  If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

area
: specifies the address of the work area being presented to the control program.

  If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

The address of an input buffer segment containing the next logical record is returned to the user in parameter register 1 after the macro-instruction is executed. The control program returns the record length in the logical record length (DCBLRECL) field of the data control block.

CAUTIONS: Substitute-mode operation can be used only with exchange buffering, and with all formats except variable-blocked records. When substitute mode is requested and the combination of channels, CPU, and input/output devices cannot support the request, move-mode operation and simple buffering are used. (Refer to Table 9.)

When chained scheduling is used with format-U records, the maximum record length is placed in the DCBLRECL field. (The work area must be large enough to contain the maximum-length record in this case.)

EXCEPTIONAL RETURNS: Refer to the locate-mode GET macro-instruction.

PROGRAMMING NOTES: Refer to the move-mode PUTX macro-instruction for a discussion of the use of buffer segments longer than the records they contain.

PUT -- Locate Mode (R)

This PUT macro-instruction provides the address of an area within an output buffer large enough to contain an output record. The user should subsequently construct, at this address, the next sequential logical record of the output data set.

Name	Operation	Operand
[symbol]	PUT	{dcb-addrx / (1)}

dcb
: specifies the address of the data control block opened for the data set.

  If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

The address of the next buffer segment large enough to contain the output record is returned to the user in parameter register 1 after the macro-instruction is executed.

CAUTION: Before executing this macro-instruction with format-U or -V records, the user must place the length of the record in the logical record length (DCBLRECL) field of the data control block. For format-U records, the DCBRECL field determines the length of the record that is subsequently written. For format-V records, the DCBRECL field is used to locate a buffer segment of sufficient size, but the length of the record actually constructed in the segment is verified before the record is written. The length placed in the DCBRECL field can therefore be greater than the length of the format-V record that is constructed; no error will result, but the blocking factor may be affected.

This mode of the PUT macro-instruction can be used with all record formats with simple buffering. The PUTX macro-instruction cannot be used if the data control block was opened for the locate-mode PUT macro-instruction.

EXCEPTIONAL RETURNS: The synchronous error exit (SYNAD) routine specified in the data control block is given control if there is no room in the buffers to construct the next record, and writing a buffer will cause a permanent error condition.

When the SYNAD routine is given control, the general registers will be set as shown in Table 13. Status indicators are listed in Appendix G.

PROGRAMMING NOTES: The control program will write out the last buffer when the data set is closed.

PUT -- Move Mode (R)

This PUT macro-instruction moves a logical record into an output buffer.

Name	Operation	Operand
[symbol]	PUT	{dcb-addrx / (1)}, {area-addrx / (0)}

dcb
specifies the address of the data control block opened for the data set being processed.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

area
specifies the address of the next logical record to be moved into the output buffer.

If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

The control program returns the address of the work area containing the logical record in register 1. (This feature provides compatibility with the substitute-mode PUT macro-instruction, and allows move mode to be used by the control program when substitute mode cannot be supported.)

CAUTIONS: With simple buffering, all record formats may be used. With exchange buffering, all record formats except blocked format V may be used. (Refer to Table 9.) For format U records, the actual record length must be known and placed in the DCBLRECL field of the data control block before the PUT macro-instruction is executed.

EXCEPTIONAL RETURNS: The user's synchronous error exit (SYNAD) routine specified in the data control block is given control if there is no room in the buffers to move the next record and writing a buffer will cause a permanent error condition.

When the SYNAD routine is given control, the general registers will be set as shown in Table 13. Status indicators are listed in Appendix G.

EXAMPLE: In the following example, the use of a move-mode PUT macro-instruction with simple buffering is shown. The address of the next logical record to be processed is returned in register 1 following the locate-mode GET macro-instruction. The record is part of an input data set associated with the data control block INVEN. After the record has been updated within the input buffer, the move-mode PUT macro-instruction is used to move the record to an output buffer. Before the PUT macro-instruction is executed, the address of the record is placed in parameter register 0. The branch instruction is used to reenter the processing loop.

```
        .
        .
        .
AAV     GET     INVEN
        .
        .
        .
        LR      0,1
        PUT     REPORT,(0)
        B       AAV
        .
        .
        .
```

PUT -- Substitute Mode (R)

This PUT macro-instruction transfers ownership of a work area containing a logical record to the control program. In return, the ownership of a buffer segment is transferred to the user, for use as a work area. There is no movement of data in main storage.

Name	Operation	Operand
[symbol]	PUT	{dcb-addrx | (1)}, {area-addrx | (0)}

dcb
: specifies the address of the data control block opened for the data set being processed.

  If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

area
: specifies the address of the area containing the logical record that is given to the control program.

  If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

When processing format-U records, the user must place the length of the record in the logical record length (DCBLRECL) field of the data control block before executing the PUT macro-instruction.

The address of the work area presented in exchange for the record is returned to the user in parameter register 1.

CAUTION: Refer to the substitute-mode GET macro-instruction.

EXCEPTIONAL RETURNS: Refer to the locate-mode PUT macro-instruction.

EXAMPLES: In the following example, the use of the substitute-mode GET and PUT macro-instructions when a data set is being updated (and the records are of fixed lengths) is shown. The address of a work area, equal in size to a logical record, is placed in register 0. This work area is presented to the control program by the first GET macro-instruction; in return, the user receives, in register 1, the address of the area containing the first logical record to be updated. After the record has been processed, its address is placed in register 0, and the PUT macro-instruction returns the area containing the record to the control program. To complete the cycle, the control program returns the address of a work area in register 1. The user places the address in register 0, and executes an unconditional branch to the GET macro-instruction.

```
        .
        .
        .
        LA      0,WORK
QQA     GET     INDATA,(0)
        .
        .
        .
        LR      0,1
        PUT     OUTDATA,(0)
        LR      0,1
        B       QQA
        .
        .
        .
```

If the user is creating a new output data set, a similar loop can be constructed by using only the substitute mode PUT macro-instruction, as follows:

```
        LA      10,WORKAREA
EEZ     GET     REPTOLD         locate-mode
        .
        .
        .
        GET     INVENOLD        locate-mode
        .
        .
        .
        GET     SOURCE          locate-mode
        .
        .
        .
        create new record in work area, making all references
            using register 10
        .
        .
        .
        PUT     NEWDCB,(10)
        LR      10,1
        B       EEZ
```

PUTX -- Update Mode (R)

This PUTX macro-instruction returns an updated logical record to a data set. The record must have been retrieved by a locate-mode GET macro-instruction. There is no movement of data in main storage.

Name	Operation	Operand
[symbol]	PUTX	{dcb-addrx {(1)

dcb
specifies the address of the data control block opened for the data set being updated. This address must be the same as the one specified in the GET macro-instruction used to retrieve the record.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTIONS: The following cautions apply:

- The data set must reside on a direct-access device.
- The update option must have been specified in the OPEN macro-instruction that opened the data control block.
- This macro-instruction must have been preceded by a locate-mode GET macro-instruction that referred to the same data set.
- For blocked-format records, if any logical record in a block has been returned by a PUTX macro-instruction, the control program will not write the entire block back to the data set until all the logical records have been processed.
- The length of the block cannot be altered before the record is updated by this macro-instruction.
- New records cannot be inserted.

EXCEPTIONAL RETURNS: Any errors in writing back the block are discovered by the next GET macro-instruction that attempts to use the buffer. Hence, there is no unusual return directly from this macro-instruction.

EXAMPLE: In the following example, the use of a PUTX macro-instruction when records are being updated is shown. The locate-mode GET macro-instruction provides the address of the next record to be updated. The PUTX macro-instruction, after processing the record, returns it to the data set. The conditional branch instruction tests the condition code. If the record is to be updated, the next sequential instruction is executed; if it is not to be updated, another GET macro-instruction will be issued to locate the next record. The unconditional branch following the PUT macro-instruction is used to reenter the processing loop. When all the input records are processed, the EODAD routine is given control.

```
          .
          .
          .
LLS       GET DCBA
          .
          .
          .
          BH LLS
          .
          .
          .
          PUTX DCEA
          .
          .
          .
          B LLS
          .
          .
          .
```

PUTX -- Output Mode (R)

The PUTX macro-instruction causes a logical record contained in a buffer of an input data set to be written as the next sequential record of an output data set. When exchange buffering is used with both data sets, there is no movement of data in main storage.

Name	Operation	Operand
[symbol]	PUTX	{dcbout-addrx / (1)},{dcbin-addrx / (0)}

dcbout
specifies the address of the data control block opened for the output data set.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

dcbin
specifies the address of the data control block opened for the input data set.

If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

CAUTIONS: The following cautions apply:

- PUTX and locate- or substitute-mode PUT macro-instructions cannot be used on the same data set. The MACRF operand of the DCB macro-instruction for the output data set must specify move mode (M).
- The output-mode PUTX macro-instruction must be preceded by a locate-mode GET macro-instruction that refers to the input data set.
- If exchange buffering is used, a record may not be available to the programmer after it is released by the PUTX macro-instruction.

COMPATIBLE RECORD FORMATS AND BUFFERING TECHNIQUES: Normally, when the PUTX macro-instruction is used, data sets with the same record formats and buffering techniques are processed together. However, the control program supports certain variations from this procedure. Table 14 indicates which combinations of input and output record formats are acceptable. For each acceptable combination, the table indicates the buffering technique possible for the output data set. Either simple or exchange buffering can be used with the input data set.

Table 14. Acceptable Record Formats and Corresponding Buffering Techniques for QSAM and the PUTX Macro-Instruction

dcbin (locate mode) \ dcbout (move mode)	to U (1)	to F (2)	to FB (2)	to V (3)	to VB (3)
from U	SE	---	---	---	---
from F	SE	SE	SE	---	---
from FB	SE	SE	SE	---	---
from V	SE	---	---	SE	S
from VB	S	---	---	S	S

where:

S indicates that a simple buffered output data set may be used

E indicates that an exchange buffered output data set may be used

U indicates format-U records

F indicates format-F records

FB indicates format-F blocked records

V indicates format-V records

VB indicates format-V blocked records

Notes for Table 14:

1. The block size for the format-U output data set must be as large as the largest logical record size of the input data sets.

2. The logical record size for format-F and -FB records must be the same for both data sets.

3. The maximum logical record size for format-V and -VB records must correspond.

EXCEPTIONAL RETURNS: Refer to the locate-mode PUT macro-instruction.

PROGRAMMING NOTES: The PUTX macro-instruction cannot be used with format-F or -FB records if the logical record sizes of the two data sets differ. However, when exchange buffering is specified, the substitute-mode GET and PUT macro-instructions may be used to pass format-F or -FB records between data sets of unlike logical record sizes. This procedure requires that both the work areas and buffer lengths be defined as being at least as large as the largest logical record. The user can then modify records in the work area provided by the control program. If all buffer segments and work areas are defined

as being larger than the largest logical record, the rightmost positions of these fields can be used to construct chains or other arrays. The cautions that accompany the substitute-mode GET macro-instruction apply here. Note that when a buffer is subdivided by the control program, the segment lengths are equal.

RELSE -- Release an Input Buffer (R)

The RELSE macro-instruction causes the remaining contents of the current input buffer to be ignored. The next GET macro-instruction will retrieve the first logical record from the next input buffer.

Name	Operation	Operand
[symbol]	RELSE	{dcb-addrx / (1)}

dcb
specifies the address of the data control block opened for the input data set.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTION: A RELSE macro-instruction is ignored if used with unblocked records or if all records in a buffer have been processed, or if it immediately follows another RELSE macro-instruction.

TRUNC -- Truncate an Output Buffer (R)

The TRUNC macro-instruction causes the current output buffer to be regarded as filled. The next PUT macro-instruction will use the next buffer to hold a logical record.

Name	Operation	Operand
[symbol]	TRUNC	{dcb-addrx / (1)}

dcb
specifies the address of the data control block opened for the output data set.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

A TRUNC macro-instruction will be ignored if used with unblocked records or when a buffer is full, or if it immediately follows another TRUNC macro-instruction.

CAUTIONS: The TRUNC macro-instruction is meaningful only with format-F and -V blocked records. Its use with format-F blocked records means that the data set cannot be considered to contain standard blocks. When

the data set is read, the RECFM operand of the DCB macro-instruction must not contain an S.

EXCEPTIONAL RETURNS: Any error resulting from the execution of this macro-instruction will be discovered by the next PUT macro-instruction that requires the use of the buffer. There is no direct exceptional return from this macro-instruction.

CNTRL -- Control a Printer or Stacker (R)

The CNTRL macro-instruction provides stacker selection of an on-line card reader, or carriage control of an on-line printer.

Name	Operation	Operand
[symbol]	CNTRL	dcb-addrx,action-{SS|SP|SK},number-value

dcb
specifies the address of the data control block (DCB) opened for the data set being processed.

action
specifies that the controlling action to be performed is one of the following:

SS - select a stacker (the number operand values are 1 or 2).
SP - space lines on the printer (the number operand values are 1, 2, or 3).
SK - skip to a carriage control tape channel (the number operand values are 1 through 12).

number
specifies a value for the controlling action to be performed, as described in the preceding operand.

A skip to a given carriage control tape channel will cause no action if the device is already at that channel.

CAUTIONS: For stacker selection, the DCBBUFNO field of the data control block must be one. Each GET macro-instruction, except the last, must be followed by a stacker-selection CNTRL macro-instruction directed to the same device. When the data set is closed, the last card read is placed in the stacker specified by the previous CNTRL macro-instruction. The CNTRL macro-instruction need not immediately follow the GET macro-instruction.

For the printer, use of control characters precludes use of the CNTRL macro-instruction.

EXAMPLE: In the following example, the on-line printer associated with the data control block PRINTOUT will skip to channel 7 of the carriage control tape.

```
EX1     CNTRL   PRINTOUT,SK,7
```

PRTOV -- Test for Printer Carriage Overflow (R)

The PRTOV macro-instruction is used to control the page format for an on-line printer. The programmer can test channel 9 or 12 of the carriage control tape for an overflow condition.

Name	Operation	Operand
[symbol]	PRTOV	dcb-addrx,number-{9|12}[,userrtn-addrx]

dcb
: specifies the address of the data control block opened for the data set being processed.

number
: specifies which channel (9 or 12) is to be tested.

userrtn
: specifies the address of a routine that is to be given control if the overflow condition exists. If this operand is omitted, an automatic skip to channel 1 will be performed when an overflow condition is found.

EXCEPTIONAL RETURNS: The contents of the general registers upon entry to the user's overflow routine are as follows:

Register	Contents
0 and 1	Contents destroyed
2 through 13	Those that existed before the macro-instruction was executed
14	The return address
15	The address of the exit routine

EXAMPLE: In the following example, channel 9 will be tested for an overflow condition. Since the optional error routine address has been omitted, an overflow condition will cause a skip to channel 1.

```
EX1    PRTOV DCBOUT,9
```

PROGRAMMING NOTES: The test for an overflow condition is performed synchronously or asynchronously, depending on whether a user's routine has been provided:

- If a routine has been provided, the test is performed when the PRTOV macro-instruction is issued. To ensure that all printing operations will be completed before the test is made, only one output buffer should be used.

- If no routine has been provided, the test is performed just before printing the record referred to by the next PUT macro-instruction. All previous printing operations will be completed before the test is made.

An overflow condition is detectable after printing of the line that follows the line corresponding to the channel 9 or channel 12 punch in the carriage control tape. Note that if the locate mode of PUT is used, a buffer provided by one PUT macro-instruction is not written until execution of the next PUT macro-instruction.

This macro-instruction causes no action if used for a device other than a printer.

BASIC SEQUENTIAL ACCESS METHOD (BSAM)

The macro-instructions included in this group permit the user to create and gain access to blocks of a sequentially organized data set, and to create a data set that can be processed by the basic direct-access method (BDAM). The user can remain device independent by restricting himself to a subset of macro-instructions. For users to whom device independence is not a limiting factor, a more extensive set can be used.

Macro-Instruction	Function
Device Independence:	
READ	Reads a block.
WRITE	Writes a block.
CHECK	Waits and tests for completion.
CLOSE(TYPE=T)	Processes labels and repositions volumes.
PRTOV	Tests for printer carriage overflow.
Tape/Direct Access Device Independence:	
NOTE	Notes where block was written.
POINT	Repositions to a specified block.
BSP	Backspaces a block.
Device-Dependence:	
CNTRL	Controls a card reader, printer, or magnetic tape drive.
WRITE	Creates a direct organization data set; used with format-F records.
WRITE	Creates a direct organization data set; used with format-V or -U records.

The user must include a DCB macro-instruction in his program. All general service macro-instructions can be used; the OPEN and CLOSE macro-instructions must be used.

BSAM responds to control characters when logical records are written to a printer or punch. The control characters are listed in Appendix F, which also contains a discussion of SYSOUT writers.

The OPEN macro-instruction option specifying the intended method of input/output processing, opt_1, has the following effect on the BSAM macro-instructions:

OPEN Macro-Instruction Operands	Effect
INPUT	A WRITE macro-instruction may not be used.
OUTPUT	A READ macro-instruction may not be used.
INOUT,OUTIN	All macro-instructions may be used.
RDBACK	A WRITE macro-instruction may not be used.
UPDAT (direct-access device only)	A WRITE macro-instruction will replace the block last addressed by a READ macro-instruction.

DCB -- Define Data Control Block for BSAM

The DCB macro-instruction reserves space for a data control block and informs the control program of the characteristics and intended uses of a data set.

Name	Operation	Operand
[symbol]	DCB	DSORG={PS|PSU},MACRF=code [,DDNAME=symbol][,DEVD=code] [,OPTCD={W|C|WC}][,RECFM=code] [,LRECL=absexp][,BLKSIZE=absexp] [,NCP=absexp][,BUFNO=absexp] [,BFALN={F|D}][,BUFL=absexp] [,BUFCB=relexp][,EODAD=relexp] [,EXLST=relexp][,SYNAD=relexp]

The keyword operands DSORG and MACRF can be supplied by only the DCB macro-instruction. The remaining operands can be supplied after assembly time by other sources; these sources are indicated in the operand descriptions.

DSORG
specifies the organization of the data set as one of the following:

PS - a physical sequential organization
PSU - a physical sequential organization in which any data set contains location-dependent information with respect to this data set. The data set is unmovable.

Note: The DCB macro-instruction is ordinarily the only source of the DSORG operand. However, if a direct-organization data set is to be created, the DCB macro-instruction must specify DSORG=PS and the DD statement must specify DSORG=DA.

MACRF

specifies the types of macro-instructions that will be used in processing the data set, as follows:

```
                 (R[C|P])
                 (W[C|P|L])
,MACRF={ (R[C],W[C]) }
        { (R[P],W[P]) }
```

R - READ macro-instruction
W - WRITE macro-instruction
C - CNTRL macro-instruction
P - POINT macro-instruction (which implies the NOTE)
L - WRITE macro-instruction (load mode for a direct data set)

If only (R) or (W), or (R,W) is written, device-independent sequential processing is assumed. Note that the CNTRL macro-instruction cannot be used if the NOTE and POINT macro-instructions are used.

DDNAME

specifies the name of the DD statement that will be used to describe the data set to be processed.

This information can be supplied by the user's problem program before opening the data control block.

DEVD

specifies the device or devices on which the data sets may reside. For certain devices, additional keyword operands can be written to provide device-dependent information. The format of these operands is as follows:

```
                 DA[,KEYLEN=value]
                 TA[,DEN={0|1|2}][,TRTCH={C|E|T|ET}]
,DEVD={ PT[,CODE={I|F|B|C|A|T|N}]
        PR[,PRTSP={0|1|2|3}]
        {PC}[,MODE={C|E}][,STACK={1|2}]
        {RD}
```

For each device type, a device-dependent area is reserved in the data control block. The device types, from DA to RD, are listed in descending space requirements. To achieve program device independence over a limited set of devices up to the time of execution, the programmer should specify the device type requiring the largest area. If the operand is omitted, the maximum device-dependent area is reserved.

Each of the additional keyword operands is described under the optional value of the DEVD operand with which if belongs. Note that the alternate sources of information for these operands do not apply to the DEVD operand.

DA: specifies a direct-access device.

KEYLEN

specifies the length, in bytes, of the key associated with a physical block. When a record is read or written, the number of bytes transmitted equals the key length plus the block length. The maximum length of the key is 255 bytes.

In this access method, channel commands for direct-access devices are established (by the control program) to read both the key and data, if required.

This arrangement presents no problem unless the data set has been incorrectly defined; i.e., if the user had omitted the KEYLEN parameter, but the data set did in fact contain keys.

This information can be supplied by any of the three alternate sources. If the operand is written as KEYLEN=0 in the DCB macro-instruction, the alternate sources are ignored and no keys are read or written during the execution of the program.

TA: specifies a magnetic tape drive.

DEN
specifies a value for the tape recording density in bits per inch as listed in Table 15.

This information can be supplied by the DD statement or the user's problem program. If not supplied by any source, the lowest density is assumed.

Table 15. DEN Values

DEN Value	Tape Recording Density (bits/inch)		
	Model 2400		Model 7340
	7 Track	9 Track	
0	200	-	1511
1	556	-	3022
2	800	800	-

TRTCH
is used with seven-track tape to specify the tape recording technique, as follows:

C - data conversion feature is to be used. If data conversion is not available on the allocated device, only format-F and -U records are supported by the control program.
E - even parity is to be used (if omitted, odd parity is assumed).
T - BCDIC to EBCDIC translation is required.

This information can be supplied by the DD statement or the user's problem program.

PT: specifies a paper tape device.

CODE
specifies the code in which the data was punched, as follows:

I - IBM BCD perforated tape and transmission code (8 tracks)
F - Friden (8 tracks)
B - Burroughs (7 tracks)
C - National Cash Register (8 tracks)

A - ASCII (8 tracks)
T - Teletype (5 tracks)
N - no conversion (format-F records only)

This information can be supplied in the DD statement or the user's problem program. If not supplied by any source, I is assumed.

The following apply when conversion is requested:

- Characters that are deleted in the conversion process are not counted in determining the block size.
- A character determined to have a parity error will not be converted when the record is moved to the user's input area.

PR: specifies a printer.

PRTSP
specifies the line spacing on a printer as 0, 1, 2, or 3. (This operand is valid only if control characters are not specified in the DCBRECFM field of the data control block.)

This information can be supplied by the DD statement or the user's problem program. If not supplied by any source, 1 is assumed.

PC: specifies a card punch.

RD: specifies a card reader or card read punch.

MODE
can be used with a card reader, a card punch, or a card read punch and specifies the mode of operation as follows:

C - the card image (column binary) mode
E - EBCDIC code

This information can be supplied by the DD statement or the user's problem program. If not supplied by any source, E is assumed.

STACK
can be used with a card reader, a card punch, or a card read punch, and specifies which stacker bin is to receive the card. Either 1 or 2 is specified.

This information can be supplied by the DD statement or the user's problem program. If not supplied by any source, 1 is assumed.

OPTCD
specifies an optional service to be provided by the control program, as follows:

W - perform a write validity check (direct-access devices only)
C - process using the chained scheduling method
WC - perform both validity checking and chained scheduling

This information can be supplied by the DD statement or by the user's problem program. If not supplied from any source, neither service is performed.

RECFM

specifies the characteristics of the records in the data set, as follows:

```
                 U[T][A|M]
,RECFM={ V[B|T][A|M]                    }
        F[B|S|T|BS|BT|BST|ST][A|M]
```

where the record format is:

U - undefined records
V - variable-length records
F - fixed-length records

the physical attributes are:

B - blocked records
S - standard blocks; no truncated blocks or unfilled tracks within the data set, with the possible exception of the last block or track
T - track overflow is to be used

and the record contains:

A - ASA control character (Refer to Appendix F.)
M - machine code control character (Refer to Appendix F.)

Record format information can be supplied by any of the three possible alternate sources. The absence of a physical attribute mnemonic (B, S, and T) implies the opposite of that attribute.

If no record format information is supplied, a format-U record without a control character is assumed.

If A is specified for a data set on an IBM 1442 Card Read Punch, which is opened for OUTPUT, each WRITE macro-instruction causes a card to be punched and ejected into the stacker indicated by the control character. But, if the data set is opened for INOUT, WRITE macro-instructions do not cause card ejection; READ macro-instructions cause the card at the punch station to be ejected into the stacker bin indicated by the control character. In this case, it is recommended that the STACK option specify stacker bin 1, because selection of stacker bin 2 for use with the READ macro-instruction takes precedence over prior stacker selection for the card at the punch station when the READ macro-instruction is executed.

If M is specified for a data set on an IBM 1442 Card Read Punch, which is opened for OUTPUT, the system uses each control character as the command code of a channel command word (CCW). In this way, the problem program can issue write or control commands with stacker selection and with or without card ejection. In addition, the card image modifier of the write command may be specified. If stacker bin 2 is selected in a command code, the next card to be ejected is stacked in stacker bin 2, irrespective of other command codes up to and including the one causing the card ejection. If the data set is opened for INOUT, machine code control characters are used as described for OUTPUT, with the execution of the WRITE macro-instruction. Execution of a READ macro-instruction always causes the card at the punch station to be stacked. In this case,

it is recommended that the STACK option specify stacker bin 1, because selection of stacker bin 2 for use with the READ macro-instruction takes precedence over prior stacker selection for the card at the punch station when the READ macro-instruction is executed. In the WRITE macro-instruction, if the card eject modifier is used, the card at the read station cannot be retrieved because the stacking operation moves that card from the read station to the punch station.

For paper tape records, format-U records imply the use of end-of-record (EOR) characters to delimit the block. The only valid formats on paper tape are U and unblocked F.

LRECL
specifies the length, in bytes, of a format-F logical record, or the maximum length of a format-V logical record. This operand is omitted for format-U records. The maximum value is 32,760.

This information is not normally used by the basic sequential access method (BSAM) but can be entered in the DCB (or by any one of the three alternate sources) so that it will be recorded in the data set label and available to the user's problem program. If a short block is encountered when format FB blocks are being read, the control program uses the LRECL value to test for a true short block.

When paper tape is being read (with conversion), the control program places the record length in the DCBLRECL field after each READ macro-instruction.

BLKSIZE
specifies the maximum block length in bytes. For format-F records, the length must be an integral multiple of the LRECL value. For format-V records, the length must include the four-byte block-length field that is recorded at the beginning of each block. The maximum value is 32,760.

This information can be supplied by any of the three alternate sources.

When reading format-U records from paper tape, the number of characters moved to the user's input area is limited either by the number specified in the DCBBLKSI field or by the occurrence of an EOR character, whichever comes first. This could result in a movement of zero characters, indicating no data for that particular request. When reading format-F records from paper tape, the physical end of the tape must coincide with the end of a block, or a wrong length indication is given. For format-U records, the physical end of the tape is treated as an EOR character.

NCP
specifies the maximum number of READ or WRITE macro-instructions that will be issued before a CHECK macro-instruction. The maximum number that can be specified is 99; however, the number must not exceed the limit on input/output requests established during system generation.

This information can be supplied by the DD statement or the user's problem program. If not supplied from any source, a maximum of one is assumed.

NOTE: Refer to Table 16 for a list of the situations in which the following four operands (BUFNO, BFALN, BUFL, and BUFCB) are applicable.

Table 16. BSAM and BPAM Buffer Acquisition and Data Control Block Field Requirements

Usage and Characteristics		Method of Obtaining Buffer Pool			User Performs All Buffering
		Automatically by OPEN	BUILD	GETPOOL	
When Issued			In data control block exit routine or before OPEN	In data control block exit routine or before OPEN	
Result		System acquires storage and structures into buffer pool	Structures storage into buffer pool	Acquires storage and structures into buffer pool	
Data Control Block Field Requirements (to be provided no later than conclusion of data control block exit routine)	DCBBUFNO	Required	Required; user sets this field before or after BUILD execution	Ignored; GETPOOL sets this field	Must be omitted[2]
	DCBBUFCB	Must be omitted	Required; user sets this field before or after BUILD execution	Ignored; GETPOOL sets this field	Ignored
	DCBBFALN	Optional[1]	Ignored	Optional[1]	Ignored
	DCBBUFL	Optional; if omitted, field is not altered and DCBBLKSI and DCBKEYLE are used	Ignored	Ignored	Ignored
Features		Provides standard options	More than one data control block can use pool; user issues GETBUF and FREEBUF	Execution time request for storage; user issues GETBUF and FREEBUF	User supplies buffers through his own methods
Cautions		Only one data control block can use pool; must use FREEPOOL	User responsible for storage acquisition and boundary alignment; must close all data control blocks before reissuing BUILD	Only 1 data control block can use pool; must use FREEPOOL	User responsible for boundary alignment

[1] If omitted, field is not altered, double-word alignment is assumed.
[2] If the DD statement is shared, the system may allocate storage for buffers specified in other data control blocks. To prevent such allocation, DCBBUFNO must be cleared to binary zeros by the data control block exit routine.

BUFNO
specifies the number of buffers to be assigned to the data control block. The maximum number that can be specified is 255; however, the number must not exceed the limit on input/output requests established during system generation.

This information can be supplied by the DD statement or the user's problem program.

BFALN
specifies the boundary alignment, in bytes, of each buffer, as follows:

F - the buffer starts on a full-word boundary (one that is not also a double-word boundary)
D - the buffer starts on a double-word boundary

This information can be supplied by the DD statement or the user's problem program.

BUFL
specifies the length, in bytes, of each buffer to be obtained for a buffer pool. The maximum value is 32,760.

This information can be supplied by the DD statement or the user's problem program. If no information is supplied, the control program calculates the length by using the value supplied for the BLKSIZE operand.

BUFCB
specifies the address of a buffer pool control block (i.e., the eight-byte field preceding the buffers in a buffer pool).

The only alternate source for this information is the user's problem program (e.g., the execution of a GETPOOL macro-instruction).

EODAD
specifies the address of the user's end-of-data set exit routine for input data sets. This routine is entered when the user requests a block and there are no more blocks to be retrieved. If no routine has been provided, the task is abnormally terminated.

The only alternate source for this information is the user's problem program.

EXLST
specifies the address of an exit list created by the programmer.

The format of this list is shown in Appendix D. Exit lists are required if:

- Label exits or data control block exits are used.
- A checkpoint is to be taken automatically at the beginning of each volume (except the first).

The only alternate source for this information is the user's program.

SYNAD
specifies the address of the user's synchronous error exit routine. The routine is entered if input/output errors result from an attempt to process data records.

The only alternate source for this information is the user's problem program.

CAUTION: The DCB macro-instruction must not be written within the first 16 bytes of a control section. It can be preceded by padding, constants, or instructions.

READ -- Read a Block (S)

The READ macro-instruction retrieves the next sequential block from an input data set and places it in a storage area.

Name	Operation	Operand
[symbol]	READ	decb-symbol,type-{SF|SB},dcb-addr ,area-addr,length-{'S'|value}

decb
specifies the name to be assigned to the data event control block (DECB) constructed as part of the expansion of the macro-instruction. The DECB starts on a full-word boundary and contains:

- An event control block (ECB) that is tested for completion of the read operation.
- A parameter list.
- A pointer to status indicators that are set following each read operation.

The format of the DECB is shown in Table 17.

Table 17. Format of the Data Event Control Block

Offset from DECB Name (bytes)	Field
+0	Event control block
+4	Type field
+6	Length field
+8	Data control block address
+12	Area address
+16	Pointer to status indicators

type
specifies one of the following:

SF - sequential forward reading of a physical sequentially organized data set.
SB - backward reading from a magnetic tape (format-F and -U records only)

dcb
specifies the address of the data control block opened for the data set being processed.

area
specifies the address of an area in main storage into which the block is to be read. If SF was written in the type field, this operand specifies the address of the first byte of the area; if SB was written, the address of the last byte is specified. If the data set resides on a direct-access device and the data portion of each block is preceded by a key, both the key and data will be read sequentially into the area. All keys in the data set must be of constant length, and the data portion of the block must be in format U, F, or V.

length
specifies, for format-U records, a value for the number of bytes to be transmitted (excluding the key, if present). If 'S" is written, the entire block is read. If the data set resides on a direct-access device and employs keys, the control program computes an effective length by adding the value specified in the DCBKEYLE field of the data control block to the block length.

Note: The length operand is ignored for format-F and -V records.

CAUTIONS: The READ macro-instruction returns control to the user's problem program before the actual transmission of data is completed. To determine whether the read operation has been completed, it is necessary to issue the CHECK macro-instruction before using the data transferred into the specified area. The DECB employed for a read operation should not be reused until the CHECK macro-instruction has been issued.

After a read operation has been checked, the length of a format-U block (normal scheduling), or a truncated block in a fixed-length blocked data set (normal or chained scheduling) can be determined from the count field of the status indicators whose addresses are in the data event control block. (Refer to Table 18 and Appendix G.)

EXCEPTIONAL RETURNS: Any exceptional condition arising in executing the READ macro-instruction is detected by the CHECK macro-instruction.

EXAMPLE: In the following example, a DECB will be produced as part of the in-line expansion. It will indicate that a forward read of the next block in the data set associated with the data control block INDCB should be performed using the area INAREA. The length operand was not written in this example, but would be required for format-U records.

```
EX1  READ    DECB,SF,INDCB,INAREA
```

L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B except for the following special operand requirements:

Operand	L Form	E Form
decb	required	required
type	required	required
MF	required	the operand must be written as MF=E

The operand MF=F does not require a parameter list address because the first operand, decb, is used as a pointer to a parameter list that was established by the L form of the macro-instruction.

WRITE -- Write a Block (S)

The WRITE macro-instruction transfers a block from the user's main storage area to a physical sequential data set.

Name	Operation	Operand
[symbol]	WRITE	decb-symbol,type-SF,dcb-addr,area-addr ,length-{'S'|value}

decb
: specifies the name to be assigned to the data event control block (DECB) constructed as a part of the expansion of this macro-instruction. (The READ macro-instruction for BSAM contains a full description of a DECB.)

type
: specifies SF for sequential forward writing of the block as part of the data set.

dcb
: specifies the address of the data control block opened for the data set being processed.

area
: specifies the starting address of the area in main storage that contains the block to be written. If the data set is being written to a direct-access device and the data portion of each block is to be preceded by a key, both the key and data will be written sequentially from the area. All the keys in the data set must be of constant length and the data portion of the block must be in format U, F, or V.

length
: specifies, for format-U records, a value for the number of bytes to be transmitted (excluding the key, if present). If 'S' is written, the maximum block length for the data set will be written. If the data set resides on a direct-access device and uses keys, the control program computes an effective length by adding the value specified in the DCBKEYLE field of the data control block to the block length.

  Note: The length operand is ignored for format-F and -V records.

CAUTION: The WRITE macro-instruction returns control before the actual transmission of data is completed. To determine whether a write operation has been completed, the CHECK macro-instruction must be issued. The DECB employed for the write operation and the main storage the block occupies should not be altered until the CHECK macro-instruction has been issued.

EXCEPTIONAL RETURNS: Any exceptional condition arising in executing the WRITE macro-instruction is detected by the CHECK macro-instruction.

EXAMPLE: In the following example, the proper use of a WRITE macro-instruction for format-U records is shown. A data event control block is constructed as part of the in-line macro-expansion. A write operation is to be performed from AREA to the data set defined by DCBOUT. Eight hundred data bytes are to be transmitted.

```
EX1    WRITE DECB,SF,DCBOUT,AREA,800
```

L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B except for the following special operand requirements:

Operand	L Form	E Form
decb	required	required
type	required	required
MF	required	the operand must be written as MF=E

The operand MF=E does not require a parameter list address because the first operand, decb, is used as a pointer to a parameter list that was established by the L form of the macro-instruction.

WRITE -- Update a Block (S)

The WRITE macro-instruction returns a block to a physical sequential data set residing on a direct-access device. The data set must be opened with the update option. Only the most recently read block can be updated and returned.

Name	Operation	Operand
[symbol]	WRITE	decb-symbol,type-SF,dcb-addr,area-addr ,length-{'S'|value}

decb
: specifies the name to be assigned to the data event control block (DECB) constructed as part of the expansion of this macro-instruction. (The READ macro-instruction for BSAM contains a full description of the DECB.)

type
: specifies SF for sequential forward writing of the block as part of the data set.

dcb
: specifies the address of the data control block opened for the data set being updated. The address must be the same as that specified in the dcb operand of the READ macro-instruction.

area
: specifies the starting address of an area in main storage from which the block is to be written. This address should be the same as that specified in the area operand of the READ macro-instruction. If the user decides to replace a block rather than update it, the area specified will be that of the replacement block, not the original block.

length
: specifies, for format-U records, a value for the number of bytes to be transmitted. If 'S' is written, the maximum block length for the data set will be written. This value must be the same as that specified for the length operand of the READ macro-instruction.

CAUTIONS: The WRITE macro-instruction returns control before the actual transmission of data has been completed. To determine whether a write operation has been completed, a CHECK macro-instruction must be issued. The DECB employed for the write operation and the main storage the block occupies should not be altered until the CHECK macro-instruction has been issued.

The update mode is provided only for data sets on direct-access devices. While it is not necessary to update and return each block, the sequence of operations for those blocks that are updated must be:

```
        .
        .
        .
READ    block A
        .
        .
        .
CHECK   await completion of read
        .
        .
        .
update block
        .
        .
        .
WRITE   block A
        .
        .
        .
CHECK   await completion of write
        .
        .
        .
```

Thus, only the block last read, or its replacement, can be returned to the data set.

EXCEPTIONAL RETURNS: Any exceptional condition arising in executing the WRITE macro-instruction is detected by the CHECK macro-instruction.

L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B except for the following special operand requirements:

Operand	L Form	E Form
decb	required	required
type	required	required
MF	required	the operand must be written as MF=E

The operand MF=E does not require a parameter list address because the first operand, decb, is used as a pointer to a parameter list that was established by the L form of the macro-instruction.

CHECK -- Wait for and Test Completion of Read or Write Operation (R)

The CHECK macro-instruction waits (if necessary) for the completion of a read or write operation and detects errors and exceptional conditions. Volume switching for input data sets is automatically handled. Additional space is automatically obtained for output data sets when the current space is filled and more WRITE macro-instructions have been issued.

Name	Operation	Operand
[symbol]	CHECK	{decb-addrx / (1)}

decb
: specifies the name of a data event control block (DECB) that was created as part of the expansion of a READ or WRITE macro-instruction.

  If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTION: The CHECK macro-instruction should be used to test for the completion of every read or write operation. For each data set, the CHECK macro-instruction must be issued in the same order in which the read or write operations were requested.

EXCEPTIONAL RETURNS: If the CHECK macro-instruction tests a read operation that attempted to gain access to a block when none is available, control will be passed to the end-of-data set exit. The user should normally issue a CLOSE macro-instruction for the data set.

If the CHECK macro-instruction determines that, because of an input/output error, the READ or WRITE macro-instruction did not complete correctly, control is given to the user's synchronous error exit (SYNAD) routine. The general registers are set to indicate the source of the error, and to provide the required control information, as shown in Table 18.

The data event control block (DECB) contains a pointer to a series of status indicators. These are arranged in main storage as shown in Appendix G.

The RETURN macro-instruction can be used to return to the control program from the SYNAD routine. The control program will then attempt to proceed as if the error had not occurred. For input on any device or for output on a unit record device, processing can be continued. In all other cases, the data control block should be closed and the routine should not return to the control program.

Note: If an error is detected by the CHECK macro-instruction and the user has not provided a SYNAD routine, the task is terminated.

Table 18. Register Contents Upon Entry to SYNAD Routine

Register	Bit	Usage
0	0 through 7	Not used.
	8 through 31	Address of the data event control block.
1	0	Set to 1 if error was caused by READ.
	1	Set to 1 if error was caused by WRITE.
	2	Set to 1 if error was caused by a BSP, CNTRL, or POINT macro-instruction
	3	Set to 1 if (1) error indicated by bit 0 did not prevent reading of the block, or (2) error indicated by bit 1 occurred during update of an existing block. Set to 0 if error prevented reading of block or occurred during creation of a new block.
	4	Set to 1 if request was illogical (e.g., a POINT macro-instruction referred to a block not contained in the data set).
	5	Set to 1 if an invalid character was encountered in paper tape translation.
	6 and 7	Not used.
	8 through 31	Address of the data control block associated with the data set being processed.
2 through 13		The contents that existed before the macro-instruction was executed.
14		The return address.
15		The address of the SYNAD routine.

EXAMPLE: In the following example, the CHECK macro-instruction tests for the completion of the input/output operations in the order in which they were requested. The operand field contains the name of the data event control block that was specified in the read or write request.

```
          .
          .
          .
EX1       READ  INDECB,SF,INVEN,WORK
          .
          .
          .
          CHECK INDECB
          .
          .
          .
EX2       WRITE OUTDECB,SF,MNTHRPRT,WORK
          .
          .
          .
          CHECK OUTDECB
          .
          .
          .
```

PROGRAMMING NOTES: If the CHECK macro-instruction detects an end-of-volume condition, the control program advances to the next volume.

A hardware-detected wrong-length block is not interpreted as an error by the CHECK macro-instruction if format-U records or truncated blocks of format-F records are being read. To determine the length of the block actually read, the programmer can examine the channel status word (part of the status indicators pointed to by the DECB) after issuing the CHECK macro-instruction. The first byte of a format-U record read backwards from magnetic tape can be located by the same method.

CLOSE (TYPE=T) -- Temporarily Disconnect a Data Set from Problem Program (S)

When the basic sequential access method is being used, this form of the CLOSE macro-instruction can be used to temporarily disconnect one or more data sets from the problem program. An OPEN macro-instruction must have been previously executed for each data control block specified in this form of the CLOSE macro-instruction.

When the data sets are temporarily disconnected, labels are processed and user label exits are taken, if necessary. Magnetic tape and direct-access volumes are then repositioned as specified or implied in this macro-instruction.

Name	Operation	Operand
[symbol]	CLOSE	({dcb-addr,[opt-code],}...),TYPE=T

dcb
: specifies the address of the data control block opened for the data set to be temporarily closed.

opt
: specifies the volume repositioning that is to be performed. Its values and meanings are as follows:

Code	Meaning
REREAD	positions the current volume to process the data set again.
LEAVE	positions the current volume to the logical end of the data set just processed. This value is assumed if the opt operand is omitted.

CAUTIONS: The following errors will cause the results indicated:

Error	Result
Temporarily closing a data control block that is not open.	No action
Temporarily closing a data control block that has not been opened for BSAM.	No action

Temporarily closing when the dcb operand does not specify the address of a data control block.	Unpredictable

PROGRAMMING NOTES: Any number of data control block addresses and associated options may be specified in the first operand field of this macro-instruction. This facility makes it possible to close data control blocks and their associated data sets in parallel.

After this macro-instruction has been executed, the user's program can issue other macro-instructions directed toward processing the data set because the data control block remains in the OPEN status.

For magnetic tape, positioning will vary, depending on the options chosen in the OPEN and CLOSE (TYPE=T) macro-instructions. Table 19 defines a position number for labeled and unlabeled tapes and Table 8 relates the options in the macro-instructions to the repositioning of the tape volumes.

Table 19. Magnetic Tape Temporary Positions - BSAM

Position	Labeled and Unlabeled Tape
1	Preceding first data block of the data set
2	Preceding tape mark that terminates last data block of the data set

The parameter list resulting from expansion of the CLOSE (TYPE=T) macro-instruction contains a full-word entry for each data control block with its associated options. The three low-order bytes of each word contain the 24-bit address of a data control block. The high-order byte contains a code, as follows:

Bit	Binary Contents	Meaning
0	0	Another parameter follows
0	1	Last entry in list
1	--	(Reserved)
2-3	00	Use DD control statement disposition
	01	Position volume for REREAD
	11	Position volume for LEAVE
4-7	--	(Ignored)

NOTE -- Provide Position Feedback (R)

The NOTE macro-instruction is used to request the relative position within a volume of the block just read or written.

Name	Operation	Operand
[symbol]	NOTE	{dcb-addrx / (1)}

dcb
specifies the address of the data control block opened for the current operation.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTIONS: All read or write requests must be checked for completion before the NOTE macro-instruction is executed. The block identification provided will always be within the current volume.

For a data set on magnetic tape, the NOTE macro-instruction should not be issued for a data set on an unlabeled volume or a volume containing nonstandard labels, if the volume was opened with either of the following conditions:

- DD statement disposition subparameter of MOD.
- OPEN macro-instruction operand of RDBACK.

PROGRAMMING NOTES: Following execution of the NOTE macro-instruction, the block identification of the last block read or written is placed in parameter register 1 by the control program.

Following the execution of this macro-instruction, the feedback information found in register 1 can be used in the POINT macro-instruction that precedes a read or write operation. The exact form of this block identification depends on whether magnetic tape or a direct-access device is used.

Magnetic Tape: If magnetic tape is used, the block identification is a 4-byte block count of the form zzCC, where

zz = binary zero bytes
CC = the block number (binary) within the volume

The block identification can be used in the POINT macro-instruction to reposition the magnetic tape to the location of the block. The block identification can be used unchanged if the operation that follows the POINT macro-instruction takes place in the same direction as the operation that preceded the NOTE macro-instruction. If the directions of the two operations are not the same, the block number must be increased by one if the original direction was forward, and decreased by one if the original direction was backward.

Direct-Access Device: If a direct-access device is used, the block identification is a 4-byte value of the form TTRz, where

TT = the track number relative to the beginning of the data set on the current volume (first track equals zero)
R = the block number on that track (first block equals one)
z = the binary zero byte

If the last operation was a write operation, an additional parameter is provided by NOTE in register 0 in the form zzLL, where

zz = the binary zero bytes
LL = the number (in binary) of bytes remaining on that track

POINT -- Position to a Block (R)

The POINT macro-instruction is used to alter the sequential processing of a data set by requiring that the next read or write operation involve a specified block within the current volume.

Name	Operation	Operand
[symbol]	POINT	{dcb-addrx | (1)},{loc-addrx | (0)}

dcb
: specifies the address of the data control block opened for the data set being processed.

  If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

loc
: specifies the starting address of a four-byte field containing a block identification. The field must start on a full-word boundary.

  If (0) is written, the address of the block identification must have been loaded into parameter register 0 before execution of this macro-instruction.

  For the first block of a data set, the block identification must be the hexadecimal number 00000001. For blocks other than the first, the format of the block identification field depends on the device type; refer to the NOTE macro-instruction for full details.

CAUTIONS: All read or write operations must be checked for completion before the POINT macro-instruction is executed. The user must make sure that the block identification previously provided by a NOTE macro-instruction, and now being used in the POINT macro-instruction, refers to the same volume.

For a data set on magnetic tape, the POINT macro-instruction should not be issued for a data set on an unlabeled volume or a volume containing nonstandard labels, if the volume was opened with either of the following conditions:

- DD statement disposition subparameter of MOD.
- OPEN macro-instruction operand of RDBACK.

EXCEPTIONAL RETURNS: The execution of a POINT macro-instruction results in an error if a volume cannot be properly repositioned or if an invalid block identification is specified. Such an error causes the next read or write operation to be completed unsuccessfully and, on execution of a CHECK macro-instruction, causes control to be given to the user's synchronous error exit (SYNAD) routine.

On entry to the SYNAD routine, the contents of general registers are as shown in Table 18, and status indicators are as defined in Appendix G. On return from the SYNAD routine, the control program clears all error indicators and attempts to resume processing.

EXAMPLE: In the following example, the POINT macro-instruction is used to present a block identification to the control program so that the next read operation will retrieve the block. The NOTE macro-instruction provides the block identification following a WRITE and CHECK macro-instruction.

```
         .
         .
         .
WRITE    OUTDECB,SF,MYDCB,(4)
         .
         .
         .
CHECK    OUTDECB
FREEBUF  MYDCB,4
         .
         .
         .
NOTE     MYDCB
ST       1,SAVE
         .
         .
         .
GETBUF   MYDCB,4
POINT    MYDCB,SAVE
READ     INDECB,SF,MYDCB,(4)
         .
         .
         .
```

PROGRAMMING NOTES: The following considerations apply to data sets on direct-access devices:

- A WRITE macro-instruction following a POINT macro-instruction over writes the block identified in the POINT macro-instruction. The NOTE macro-instruction returns the identification field of the block just written. To reposition so that writing will begin at the next block, that field must be incremented by a binary one. The field must be so incremented before the next POINT macro-instruction is executed.
- If the POINT macro-instruction is used in the UPDAT mode, a READ macro-instruction must be issued following the POINT macro-instruction.
- The POINT macro-instruction has the effect of a NOP instruction if used in processing an input stream or SYSOUT data set on a magnetic tape or unit record device.

BSP -- Backspace a Block (R)

The BSP macro-instruction backspaces a block on the current magnetic tape or direct-access volume. Backspacing is always toward the load point (or the equivalent on direct-access) regardless of the OPEN macro-instruction's parameters or the direction of reading.

Name	Operation	Operand
[symbol]	BSP	{dcb-addrx {(1)

dcb
specifies the address of the data control block opened for the data set to be backspaced.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTIONS: All read or write operations must be checked for completion before the BSP macro-instruction is executed.

The BSP macro-instruction must not be issued if the data set was written on a direct-access device using the track overflow feature.

The BSP macro-instruction must not be issued if a CNTRL, NOTE, or POINT macro-instruction is being used on this data set.

EXCEPTIONAL RETURNS: The BSP macro-instruction will return control to the program if a tape mark or the beginning of extent on the current volume is encountered.

Following execution of the BSP macro-instruction, register 15 contains binary zero if the operation completed normally. It contains binary four if the operation did not complete normally.

PRTOV -- Test for Printer Carriage Overflow (R)

The PRTOV macro-instruction is used to control the page format for an on-line printer. The programmer can test channel 9 or 12 of the printer control tape to determine if an overflow condition exists.

Name	Operation	Operand
[symbol]	PRTOV	dcb-addrx,number-{9|12}[,userrtn-addrx]

dcb
specifies the address of the data control block opened for the data set being processed.

number
specifies either 9 or 12 as the channel to be tested for an overflow condition.

userrtn
specifies the address of a routine that is to be given control if an overflow condition exists. If this operand is omitted, an automatic skip to channel 1 will be performed when an overflow condition is found.

EXCEPTIONAL RETURNS: The contents of the general registers upon entry to the user's overflow routine are as follows:

Register	Contents
0 and 1	Contents destroyed
2 to 13	Those that existed before the macro-instruction was executed
14	The return address
15	The address of the overflow routine

PROGRAMMING NOTES: The test for an overflow condition is performed synchronously or asynchronously, depending on whether a user's routine has been provided:

- If a routine has been provided, the test is performed when the PRTOV macro-instruction is issued. The CHECK macro-instruction should be used to ensure that all printing operations are successfully completed before PRTOV is issued. If printing operations are not complete, the PRTOV macro-instruction will wait for their completion before testing for an overflow condition, but will not test for errors or exceptional conditions.
- If no routine has been provided, the test is performed just before the record referred to by the next WRITE macro-instruction. All previous printing operations will be completed before this test is made.

An overflow condition is detectable after printing of the line that follows the line corresponding to the channel 9 or channel 12 punch in the carriage control tape.

This macro-instruction causes no action if used for a device other than a printer.

EXAMPLES: In EX1, an overflow condition on channel 9 of the printer control tape will result in an automatic skip to channel 1 since the operand, userrtn, has been omitted. In EX2, an overflow condition on channel 12 will result in control being given to the routine OVERFLOW.

```
EX1  PRTOV   OUTDCB,9
EX2  PRTOV   PRINTDCB,12,OVERFLOW
```

CNTRL -- Control On-Line Input/Output Devices (R)

The CNTRL macro-instruction is used to control magnetic tape drives and on-line card readers and printers.

Name	Operation	Operand
[symbol]	CNTRL	dcb-addrx,action-code[,number-value]

dcb
specifies the address of the data control block opened for the data set being processed.

action
specifies the requested operation, as follows:

Code	Result
SS	causes a stacker selection for a card reader.
SP	causes a line space on a printer.
SK	causes a skip on the carriage control tape for a printer.
BSR	causes a backspace over a specified number of blocks on magnetic tape.
BSM	causes a backspace past a magnetic tape mark and a forward space over the tape mark.
FSR	causes a forward space over a specified number of blocks on magnetic tape.
FSM	causes a forward space past a magnetic tape mark and a backspace over the tape mark.

Note: For magnetic tape, BSR and BSM mean toward the load point (that is, the physical beginning of the tape); FSR and FSM mean toward the end of the tape in a forward direction.

number
specifies a value for the stacker, number of lines, carriage tape channel, or number of blocks on magnetic tape to qualify the action operand. The maximum value is 32,767. Depending upon the action code selected, the number operand is either required, optional, or not allowed.

action code	number operand
SS	required; specify as 1 or 2
SP	required; specify as 1, 2, or 3
BSR	optional; one block assumed if operand omitted
BSM	not allowed
FSR	optional; one block assumed if operand omitted
FSM	not allowed
SK	required; specify as 1 through 12

CAUTIONS: Read and write operations must be checked for completion before the CNTRL macro-instruction is issued.

For card readers, stacker selection is made by issuing the CNTRL macro-instruction after each read operation except the last. The CNTRL macro-instruction must not be issued after the last read operation; the last card is automatically stacked with the previous card when the data set is closed. If minimum device dependence is desired, the DCBSTACK field of the data control block must specify stacker 1.

For printers, a skip to a given channel will result in no action if the device is already at that channel.

The use of control characters precludes the use of the CNTRL macro-instruction.

EXCEPTIONAL RETURNS: When magnetic tape is used, unsatisfactory completion of this macro-instruction will cause control to be passed to the user's synchronous error exit (SYNAD) routine. The general register contents are provided in Table 18. Status indicators are shown in Appendix G. Control will be returned to the user if a tape mark is encountered while an attempt is being made to forward space or backspace blocks. (Control is not given to the SYNAD routine.) Register 15 contains zero if the operation completed normally; otherwise, it contains a count of the remaining number of forward spaces or backspaces that were not completed.

WRITE -- Create a Direct Organization Data Set - Format-F Records (S)

The WRITE macro-instruction is used to add a block to a direct organization data set being created for the user by BSAM. The data set can be subsequently processed by the basic direct access method (BDAM). The use of this macro-instruction precludes the use of other BSAM macro-instructions (except CHECK).

Name	Operation	Operand
[symbol]	WRITE	decb-symbol,type-{SF|SD},dcb-addr ,area-addr

decb
: specifies the name to be assigned to the data event control block (DECB) constructed as part of the expansion of this macro-instruction. (Refer to the READ macro-instruction for a full description of the DECB.)

type
: specifies a value for the type of block to be written, as follows:
SF - new data block
SD - dummy data block

dcb
: specifies the address of the data control block opened for the data set being processed. The DCB macro-instruction must have specified the load mode of operation (by an L in the MACRF operand).

area
: specifies the starting address of the area in main storage that contains the block to be written. If the user has specified that keys are employed, the key and data are written sequentially from the area. All keys must be the same length.

  In a direct organization data set consisting of format-F records, all tracks must be filled. The user can request that the control program write a dummy block to aid in the following:

  - Spacing blocks so that the user's method of retrieval by relative block or track number will locate the current block.

- Reserving track space for future additions at, or near, the desired block.

When keys are employed, a dummy block is defined to have the first byte of the key set to FF (hexadecimal) and the first byte of the data area set to the block sequence number (identification) on the track. The control program provides both of these values when the block is written. For this reason, the area operand need only specify an area equal to the key length plus one byte. When new data blocks are subsequently added to the existing direct organization data set, the control program recognizes the dummy block and replaces it with a data block.

When keys are not employed, the user must define a dummy block to suit his needs, and be prepared to recognize the dummy block when adding a new block to an existing direct organization data set.

CAUTIONS: The WRITE macro-instruction returns control before the actual transmission of data has been completed. To determine whether a write operation has been completed, a CHECK macro-instruction must be issued. The DECB employed for the write operation and the main storage the block occupies should not be altered until the CHECK macro-instruction has been issued.

Each track allocated to the data set must be initialized. The user can choose either to continue writing dummy blocks until all tracks are initialized or to specify the release (RLSE) option in the DD statement. The latter will release all unused tracks when the data control block is closed.

EXCEPTIONAL RETURNS: The following codes will be placed in register 15 following execution of the WRITE macro-instruction:

Code (hexadecimal)	Interpretation
00	Block will be written - there is more space on the current track.
04	Block will be written - the current track will then be full.
08	Block will be written - the next WRITE must be followed by CHECK to cause execution of end-of-volume procedures.
12	Block will not be written - CHECK must be issued for previous WRITE before reissuing this request.

L- AND E-FORM USE: Refer to the READ macro-instruction for details.

PROGRAMMING NOTES: The control program will write record zero on the current track when it is filled, and will advance to the next track.

WRITE -- Create a Direct Organization Data Set - Format-U or -V Records or a Capacity Record (S)

The WRITE macro-instruction is used to add a block to a direct organization data set being created for the user by BSAM. The data set can subsequently be processed by the basic direct access method (BDAM). The use of this macro-instruction precludes the use of other BSAM macro-instructions (except CHECK).

Name	Operation	Operand
[symbol]	WRITE	decb-symbol,type-{SF|SZ},dcb-addr ,area-addr[,length-{'S'|value}]

decb
specifies the name to be assigned to the data event control block (DECB) constructed as part of the expansion of this macro-instruction. The READ macro-instruction contains a full description of the DECB.

type
specifies a value for the type of block to be written, as follows:

SF - new data record
SZ - capacity record (i.e., record zero). The control program supplies the data, writes a capacity record, and advances to the next track.

dcb
specifies the address of the data control block (DCB) opened for the data set being processed. The DCB macro-instruction must have specified the load mode of operation (by an L in the MACRF operand).

area
for data blocks, this operand specifies the starting address of the area containing the block to be written. If the user has specified that keys are employed, the key and data are written sequentially from the area. (All keys must be the same length and the data portion of the block must conform to the standard format for U or V records.) For capacity records, this operand is ignored and can be omitted.

length
for format-U records, this operand specifies a value for the number of data bytes to be written. (The control program will add the key length to this value.) If the maximum length data block is to be written, the user can write 'S'.

This operand is ignored and can be omitted when format V data records or capacity records are being written.

CAUTIONS: The WRITE macro-instruction returns control before the actual transmission of data has been completed. To determine whether a write operation has been completed, a CHECK macro-instruction must be issued. The DECB employed for the write operation and the main storage the block occupies should not be altered until the CHECK macro-instruction has been issued.

If the user attempts to write too many blocks on a track, the control program will ignore the write request and return a code in register 15. (Refer to "Exceptional Returns.") The user must write a capacity record to advance to the next track.

Each track allocated to the data set must be initialized. The user can choose either to continue writing capacity records until all the tracks are initialized, or to specify the release (RLSE) option in the DD statement. The latter will release all unused tracks when the data control block is closed. If too many capacity records are written, the control program will request additional tracks for the data set, provided that secondary allocation has been requested; it will then continue without notifying the user.

Track overflow cannot be used with format-U or -V records. The control program does not automatically advance tracks.

EXCEPTIONAL RETURNS: After execution of the WRITE macro-instruction, the following return codes are placed in register 15:

Code (hexadecimal)	Interpretation
00 (Type SZ)	Capacity record will be written - another track is available.
00 (Type SF)	Block will be written - there is more space on the current track.
04	No room on current track to write block; user must write capacity record before reissuing this request.
08	Capacity record will be written - the next WRITE must be followed by CHECK to cause execution of end-of-volume precedures.
12	Block will not be written - CHECK must be issued for previous WRITE before reissuing this request.

L- AND E-FORM USE: Refer to the READ macro-instruction.

BASIC PARTITIONED ACCESS METHOD (BPAM)

The DCB, FIND, BLDL, and STOW macro-instructions are used with the partitioned data organization, in addition to the general service macro-instructions instructions (BUILD, GETPOOL, FREEPOOL, GETBUF, FREEBUF, OPEN, and CLOSE), and the BSAM macro-instructions (READ, WRITE, CHECK, POINT, and NOTE).

Macro-Instruction	Function
FIND	Positions to first block of a data set member.
BLDL	Builds a list for use in locating data set members.
STOW	Manipulates the directory.

The OPEN macro-instruction option specifying the intended method of input/output processing, opt_1, has the following effect on the BPAM macro-instructions:

OPEN Macro-Instruction Operand	Effect
INPUT	A WRITE macro-instruction cannot be used.
OUTPUT	It is not possible to overwrite any portion of an existing member.

Partitioned Data Organization

The partitioned data organization resembles the physical sequential data organization with two significant differences:

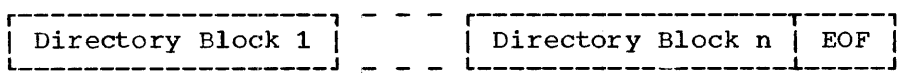
- The data set is divided or partitioned into named, retrievable members or groups of blocks. Each member is terminated with an end-of-data mark or indicator.
- The data set contains a directory that points to the first block of each member. The FIND macro-instruction uses the directory information to locate a requested member.

The partitioned data organization is supported only for direct-access devices. The entire data set must reside on one volume.

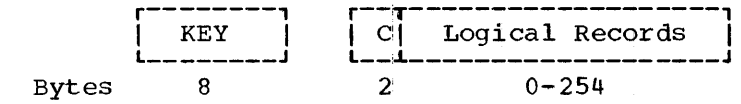
Partitioned Organization Directory Format

The partitioned organization directory occupies the beginning of the extent allocated to the data set on a direct-access device. It is searched and maintained by use of the FIND and STOW macro-instructions. The directory consists of variable-length logical records arranged in ascending order according to the binary value of the member name or alias.

The directory records are blocked into 256-byte blocks, each containing as many complete records as will fit in a maximum of 254 bytes. All remaining bytes in each block are unused and ignored. The format of a series of directory blocks is as follows:



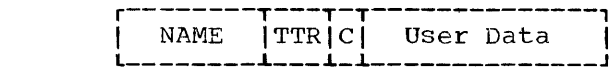
Each directory block is preceded by a hardware-defined key field and contains a count field followed by the logical records. The last usable block in the directory is followed by an end of data set indicator. The directory block format is as follows:



KEY
: specifies the name of the logical record with the highest binary value within the block. The last active block of the directory has a key of maximum binary value.

C
: specifies the number of active record bytes in the block.

Each logical record in a directory block contains a name, TTR, and count field. It may also contain a user data field. The last logical record in the last active directory block has a name field of maximum binary value. The logical record format is as follows:



Bytes	8	3	1	0-62

NAME
specifies the member name or alias. It contains up to eight alphameric characters, is left-justified, and is padded with blanks if necessary.

TTR
is a pointer to the first block of the named member; TT is the relative track from the beginning of the data set, and R is the block number on that track.

C
specifies the number of half-words contained in the user data field. It may also contain additional information about the field. The C field format is as follows:

0
when set to 1, indicates that the NAME is an alias.

1-2
specifies the number of pointers to the locations within the member that are contained in the user data field. A maximum of three pointers is allowed. Pointers so used will be automatically updated if the data set is moved or copied by an IBM utility program. The data set must be marked "unmovable" if one of the following applies:

- More than three pointers are used in the user data field.
- The pointers in the user data field or a note list do not conform to the standard format.
- The pointers are not placed first in the user data field.
- Any direct-access addresses (absolute or relative) are embedded in any data blocks or in another data set that refers to this data set.

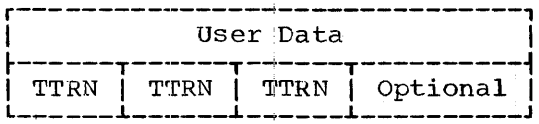
The binary code is as follows:

00 = none
01 = one
10 = two
11 = three

3-7
contains a binary value indicating the number of half-words of user data. This count must include the space occupied by any TTRN entries.

User Data
contains variable user data provided as input to the STOW macro-instruction. If pointers to locations within the member are

provided, they must be four bytes long and placed first in the user data field. The pointers must be arranged in ascending order according to their binary value. The user data field format is as follows:

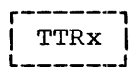


TT
is the relative track from the beginning of the data set.

R
is the block number on that track.

N
specifies, by a binary value other than zero, the number of additional pointers contained in the note list identified by the TTR. If this TTR is not a pointer to a note list, N=0.

A note list consists of additional pointers to blocks within the same member of a partitioned data set. If the note list option was indicated in the user data field, the note list will be automatically updated when the data set is moved or copied by an IBM utility program. Each entry in the note list is four bytes long. The note list entry format is as follows:



TT is the relative track from the beginning of the data set.
R is the block number.
x is available for any use.

CAUTIONS: The following considerations and restrictions apply to the use of a note list:

1. The note list must be indicated by a value in N.
2. The user must use the NOTE macro-instruction or provide entries in the exact format defined by the NOTE macro-instruction.
3. If the note list is to be maintained by an IBM utility program, each entry in the note list must be in ascending order and must point to a location with a block identification lower than the list itself, but higher than that of any previous note list.
4. Format-U records must be specified for the data set.
5. The user must use the WRITE macro-instruction to preserve the note list. The length operand of the WRITE macro-instruction must contain a value that is four times the number of entries in the list. This value must be followed by a CHECK and a NOTE macro-instruction. The NOTE macro-instruction will return the block identification of the note list; it must be preserved by the user for the STOW macro-instruction.

DCB -- Define Data Control Block for BPAM

The DCB macro-instruction reserves space for a data control block and informs the control program of the characteristics and intended uses of a data set.

Name	Operation	Operand
[symbol]	DCB	DSORG={PO|POU},MACRF=code[,DDNAME=symbol} [,OPTCD={W|C|WC}][,RECFM=code][,LRECL=absexp] [,BLKSIZE=absexp][,NCP=absexp][,KEYLEN=absexp] [,BUFNO=absexp][,BFALN={F|D}][,BUFL=absexp] [,BUFCB=relexp][,EODAD=relexp][,EXLST=relexp] [,SYNAD=relexp]

The keyword operands DSORG and MACRF can be supplied by only the DCB macro-instruction. The remaining operands can be supplied after assembly time by other sources; these sources are listed in the operand descriptions.

DSORG
specifies the organization of the data set as one of the following:

PO - partitioned organization
POU - partitioned organization in which any data set contains location-dependent information with respect to the data set. The data set is unmovable.

MACRF
specifies the type of macro-instructions that will be used to process the data set, as follows:

$$,\text{MACRF}=\left\{\begin{matrix}(\text{R})\\(\text{W})\\(\text{R,W})\end{matrix}\right\}$$

R - READ macro-instruction (implies NOTE and POINT)
W - WRITE macro-instruction (implies NOTE)

DDNAME
specifies the name of the DD statement that will be used to describe the data set to be processed.

This information can also be supplied by the user's problem program before opening the data control block.

OPTCD
specifies an optional service to be performed by the control program, as follows:

W - perform a write validity check
C - using the chained scheduling method process
WC - perform a validity check and use chained scheduling

This information can be supplied by the DD statement or the user's problem program. If not supplied by any source, none of the services are performed.

RECFM
specifies the characteristics of the records in the data set as follows:

$$,\text{RECFM}=\left\{\begin{matrix}\text{U[T]}\\ \text{V[B|T|BT]}\\ \text{F[B|T|BT]}\end{matrix}\right\}$$

where the record format is:

U - undefined records
V - variable length records
F - fixed length records

the physical attributes of the data set are:

B - blocked records
T - track overflow is to be used

Record format information (U, V, and F) can be supplied by any of the three possible alternate sources. The absence of any physical attribute mnemonic (B and T) implies the opposite of that attribute. If the record format is not supplied, a format-U record without a control character is assumed.

LRECL
specifies the length, in bytes, of a logical record. This operand applies only to format-F records. The maximum record length is 32,760 bytes.

This information can be supplied by any of the three alternate sources.

BLKSIZE
specifies the maximum block length in bytes. The maximum value is 32,760.

This information can be supplied from any of the three alternate sources.

NCP
specifies the maximum number of READ or WRITE macro-instructions that will be issued before a CHECK macro-instruction. The maximum number that can be specified is 99; however, this number must not exceed the limit on input/output requests established during system generation.

This information can be supplied by the DD statement or the user's problem program. If the information is not specified, a maximum of one is assumed.

KEYLEN
specifies the length, in bytes, of the key associated with a block. When a block is read or written, the number of bytes transmitted equals the key length plus the block length.

This information can be supplied by any of the three alternate sources.

NOTE: Refer to Table 16 for a list of the situations in which the following four operands (BUFNO, BFALN, BUFL, and BUFCB) are applicable.

BUFNO
specifies the number of buffers to be assigned to the data control block. The maximum number that can be specified is 255; however, the number must not exceed the limit on input/output requests established during system generation.

This information can be supplied by the DD statement or the user's problem program.

BFALN
specifies the word-boundary alignment, in bytes, of each buffer, as follows :

F - the buffer starts on a full-word boundary (one that is not a double-word boundary)
D - the buffer starts on a double-word boundary

This information can be supplied by any of the three alternate sources.

BUFL
specifies the length, in bytes, of each buffer to be obtained for a buffer pool. The maximum value is 32,760.

This information can be supplied by the DD statement or the user's problem program. If it is not supplied, the control program calculates the length by using the value supplied for the BLKSIZE and KEYLEN operands.

BUFCB
specifies the address of a buffer pool (i.e., the eight-byte field preceding the buffers in a buffer pool).

The only alternate source for this information is the user's problem program.

EODAD
specifies the address of the user's end-of-data set exit routine for input data sets. This routine is entered when the user requests a block in the member when there are no further blocks to be retrieved. The user may:

- Continue processing the next sequential member without repositioning.
- Reposition using a FIND or POINT macro-instruction, and continue processing another member.
- Issue a CLOSE macro-instruction.

If no routine has been provided, the task is abnormally terminated.

The only alternate source for this information is the user's problem program.

EXLST
specifies the address of an exit list created by the programmer. The format of the list is shown in Appendix D.

Exit lists are required if data control block exit routines are used.

The alternate source for this information is the user's problem program.

SYNAD
specifies the address of the user's synchronous error exit routine. The routine is entered if input/output errors result from an attempt to process data records.

The only alternate source for this information is the user's problem program.

PROGRAMMING NOTE: The BPAM DCB macro-instruction can be written at any point within a control section.

FIND -- Position to Member of Partitioned Data Set (R)

The FIND macro-instruction causes the address of the first block of a specified partitioned data set member to be placed in the indicated data control block. A subsequent read request will use that address to retrieve the blocks of the member.

Name	Operation	Operand
[symbol]	FIND	{dcb-addrx | (1)},{area-addrx | (0)},type-{D|C}

dcb
specifies the address of the data control block opened for the data set being processed.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

area
specifies the address of an area containing either the name of the member or the address of an entry for that member. The entry is in a main storage list constructed by the programmer with a BLDL macro-instruction. The format of the area is dependent upon the type field.

If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

type
specifies the information in the area parameter as one of the following:

D specifies that the directory of the partitioned data set is to be scanned for a match to the name provided in the area. The area must be eight bytes long and contain a left-justified alphameric name.

C specifies that the area points to the first byte of an area containing a relative address list. The area must be four bytes long and contain TTRK, where:

TT = the relative track number
R = the block number on track TT

K = the concatenation number[1] of the data set provided as a return from the BLDL macro-instruction

CAUTIONS: All previously issued read or write requests must have been checked before the FIND macro-instruction is issued.

EXCEPTIONAL RETURNS: After execution of the FIND macro-instruction (type D), bits 24 through 31 of register 15 contain one of the following (hexadecimal) codes to indicate the status of the operation:

Code (Hexadecimal)	Interpretation
00	Successful completion.
04	The named item was not found.
08	A permanent input/output error was detected when an attempt was made to search the directory.

Execution of the FIND macro-instruction (type C) results in the same exceptional returns as that of the POINT macro-instruction.

BLDL -- Build List (R)

The BLDL macro-instruction causes member addresses and optional information from a partitioned data set directory to be placed in a specified list constructed by the programmer in main storage.

Name	Operation	Operand
[symbol]	BLDL	{dcb-addrx / (1)},{list-addrx / (0)}

dcb
specifies the address of the data control block opened for the data set being processed. If a decimal zero is specified for the address, or if register 1 contains binary zeros, the control program searches the job library and link library.

If (1) is written, the address or binary zeros must have been loaded into parameter register 1 before execution of this macro-instruction.

list
specifies the address of the user's build list.

If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

PROGRAMMING NOTES: The format of the build list must be similar to that of the directory, and must include the names of any members for which the BLDL macro-instruction is to provide control information. Names

---

[1]When the user concatenates a partitioned data set and issues a BLDL macro-instruction, the data sets are searched in the order specified in the DD statement. K indicates the binary value of the number of the data set in which the member was found. The publication IBM Operating System/360: Job Control Language describes the concatenation of data sets.

must be placed in the list by the programmer in the same order in which they appear in the directory. The build list consists of two parts; the list description field and the list entries.

List Description Field: The description field is two half-words long and contains FFLL, where;

FF = the number of entries in the list
LL = an even number that indicates the length of each entry in bytes

List Entries: Each entry starts on a half-word boundary and contains LL bytes (a minimum of 14). The entries must be arranged by the user in ascending order, according to the binary value of the name. Each list entry is in the following format:

Bytes 8 4 1 1 C half-words

NAME
is a member name. It is left-justified and padded with blanks if less than eight bytes long. NAME must be supplied by the user.

TT
is the relative track number.

R
is the block number on that track.

K
specifies the concatenation number. If the data set was the first, or only, data set, then K=0.

z
is normally a zero byte (binary) inserted to maintain half-word boundaries. If the dcb operand was specified as zero, this byte will contain a 1 if the name was found in the link library, and a 2 if the name was found in the job library.

C
describes the user data field. Bits 3-7 indicate the number of half-words contained in the user data field.

User Data
contains as much user data from the directory entry as will fit in the remainder of the field.

The NAME or TTRK portion of a list entry can be used by a FIND macro-instruction. The area operand of the FIND macro-instruction should point to the NAME portion if the type operand is D, or to the TTRK portion if the type operand is C.

TTRKzC are retrieved from the directory and placed in the user's list.

CAUTIONS: All previously issued read or write requests must be checked before the BLDL macro-instruction is issued. The BLDL macro-instruction may refer to a data control block that specifies the execute channel program (EXCP) macro-instruction.

EXCEPTIONAL RETURNS: After execution of the BLDL macro-instruction, bits 24 through 31 of register 15 contain one of the following (hexadecimal) codes to indicate the status of the operation:

Code (hexadecimal)	Interpretation
00	Successful completion.
04	The list could not be filled. The R field of each unfilled entry will contain hexadecimal 00.
08	A permanent input/output error was detected when an attempt was made to search the directory.

STOW -- Manipulate Partitioned Data Set Directory (R)

The STOW macro-instruction causes a partitioned data set member name to be changed, added, deleted, or replaced in the data set directory. It can also be used to store additional information in the directory.

Name	Operation	Operand
[symbol]	STOW	{dcb-addrx | (1)},{area-addrx | (0)}[,type-{A|R|D|C}]

dcb
: specifies the address of the data control block opened for the data set being processed.

  If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

area
: specifies the address of an area in main storage constructed by the programmer. The exact contents of the area will depend on the type specified.

  If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

type
: specifies the type of information to be used in manipulating the data set directory, as follows:

  A (add) - the name of a member and possibly additional information is to be added to the directory. An end-of-data mark is written in the data area unless the alias option is used.
  R (replace) - a member is to be replaced in the data set. If the member to be replaced does not exist, the member specified by the area operand is added to the data set. An end-of-data mark is written in the data area unless the alias option is used.
  D (delete) - the member name is to be deleted from the directory.
  C (change) - the member name is to be changed in the directory.

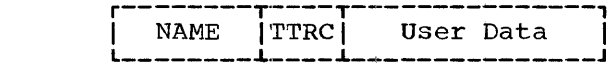
CAUTIONS: All previously issued read or write requests must be checked before the STOW macro-instruction is issued. The data control block must have been opened for output.

EXCEPTION RETURNS: After execution of the STOW macro-instruction, bits 24 through 31 of register 15 contain one of the following (hexadecimal) codes to indicate the status of the operation:

Code (hexadecimal)	Interpretation
00	Successful completion.
04	Name already exists in the directory (type A), or new name already exists in the directory (type C).
08	Name not in the directory (types D and R), or old name not in the directory (type C).
0C	No space left in the directory (types A and R).
10	A permanent input/output error was detected when an attempt was made to search the directory.

PROGRAMMING NOTES: The format of the area used by the STOW macro-instruction depends on the type of information it contains. The format for each type of information is described in the following paragraphs.

Types A and R: The area must be at least 12 bytes long and begin on a half-word boundary. The format is as follows:



	NAME	TTRC	User Data
Bytes	8	4	0 to 62

NAME
is an eight-byte name of the member being stowed.

TTR
is the relative track and block identification of the first block of the member. The quantity must be supplied when the alias option is used. When the alias option is not used, the control program stores the TTR in this field.

C
specifies the alias option and information about the user field of the area; its format is as follows:

Bits 0 1-2 3-7

0
is set to 1 by the user if the name being entered is an alias for a member name already in the directory.

1-2
indicate the number of TTRN fields in the user data fields.

3-7
indicate, by their binary values, the number of half-words contained in the user data field.

User Data
contains variable data supplied by the user. The data will be stored in the directory and can be retrieved by means of the BLDL macro-instruction.

Type D: The area specified must be eight bytes long. It contains the member name to be deleted.

Type C: The area specified must be 16 bytes long. The high-order eight bytes must contain the old member name, and the low-order eight bytes must contain the new member name.

QUEUED INDEXED SEQUENTIAL ACCESS METHOD (QISAM)

The queued indexed sequential access method (QISAM) is the sole means of creating an indexed sequential data set. The QISAM macro-instructions, which are completely direct-access-device oriented, permit the programmer to sequentially store and retrieve the records of an indexed sequential data set. They also provide automatic buffering and blocking/deblocking procedures as required. The load mode is used to create an indexed sequential data set and the scan mode is used to retrieve and update records.

The QISAM macro-instructions (DCB, GET, RELSE, SETL, ESETL, and PUTX) and the general service macro-instructions (BUILD, GETPOOL, FREEPOOL, OPEN, and CLOSE) are used with the queued indexed sequential access method. The opt_1 operand of the OPEN macro-instruction is ignored.

Macro-Instruction	Function
DCB	Constructs a data control block for an indexed sequential data set.
GET	Gets a logical record from an indexed sequential data set.
RELSE	Causes the remaining records in an input buffer to be ignored.
SETL	Specifies the point at which sequential retrieval is to begin.
ESETL	Ends sequential retrieval.
PUT	Places a logical record in an output buffer from which it is written.
PUTX	Returns an updated logical record to an indexed sequential data set.

DCB - Define Data Control Block for QISAM - Load Mode

The DCB macro-instruction reserves space for a data control block for QISAM and informs the control program of the characteristics and intended uses of a data set. The options chosen for the load-mode DCB determine processing procedures for the scan-mode of QISAM and for BISAM.

Name	Operation	Operand
[symbol]	DCB	DSORG={IS|ISU},MACRF=(P{M|L}) [,DDNAME=symbol][,OPTCD=code] [,RECFM={V|F|VB|FB}] [,LRECL=absexp][,BLKSIZE=absexp] [,RKP=absexp][,NTM=absexp][,KEYLEN=absexp] [,CYLOFL=absexp][,BUFNO=absexp] [,BFALN={F|D}][,BUFL=absexp] [,BUFCB=relexp][,EXLST=relexp][,SYNAD=relexp]

The keyword operands DSORG and MACRF can be supplied by only the DCB macro-instruction. The remaining operands can be supplied after assembly time by other sources; these sources are indicated in the operand descriptions.

DSORG
specifies the organization of the data set as one of the following:

IS - indexed sequential organization
ISU - indexed sequential organization in which any data set contains location-dependent information with respect to this data set.

Note: For space allocation purposes, this operand must also be specified in the DD control statement.

MACRF
specifies the type of macro-instruction that will be used in processing the data set; (PM) indicates that move-mode PUT macro-instructions are to be used to add records to the data set; (PL) indicates that locate-mode PUT macro-instructions are to be used. (Only the PUT macro-instruction can be used to present records when an indexed sequential data set is being created.)

DDNAME
specifies the name of the DD statement that will be used to describe the data set to be processed.

This information can also be supplied by the user's problem program before opening the data control block.

OPTCD
specifies optional services to be provided by the control program, as follows:

W - performs a write validity check
M - create master indexes as required
Y - use cylinder overflow areas
I - use independent overflow area
L - delete option: user marks records for deletion; records so marked may be actually deleted when new records are added to the data set
R - provide reorganization criteria feedback

The programmer can choose any combination of the options.

This information can be supplied by any of the three alternate sources. If this information is not supplied by any source, only reorganization criteria feedback is provided. This feedback is maintained in three fields of the data control block, as follows:

Field	Feedback
DCBRORG1	Binary number of cylinder overflow areas that are full
DCBRORG2	Binary number of tracks remaining in the independent overflow area
DCBRORG3	Binary number of READ and WRITE macro-instructions that referred to overflow records other than the first overflow record in a chain

RECFM
specifies the characteristics of the records in the data set. One of the following is written:

V - variable-length records
F - fixed-length records
VB - variable-length blocked records
FB - fixed-length blocked records

This information can be supplied by any of the three alternate sources. If this operand is not supplied by another source, F is assumed.

LRECL
specifies the length, in bytes, of a logical record. For variable-length records, the maximum record length must be specified. For unblocked records, if the relative key position is zero, LRECL is the length (or maximum length) of the data portion of the physical record. In this case only, the key does not appear in the data portion of the physical record. The maximum value must not exceed the BLKSIZE value.

This information can be supplied by any of the three alternate sources.

BLKSIZE
specifies the maximum length, in bytes, of a block. For fixed-length records, this length must be an integral multiple of the LRECL value. For variable-length records, this length must be the maximum length.

The maximum length (plus 10 bytes) must not be greater than the number of bytes available on a track of the allocated direct-access device. (The available space varies with different device types; formulas published in the publication IBM 2841 Storage Control Unit should be consulted). It is also necessary to account for the space occupied by record zero, as described in the Introduction to this section.

Note: The 10 bytes are occupied by a link field for overflow records.

This information can be supplied by any of the three alternate sources.

RKP
specifies the relative position of the first byte of the record key within each logical record. The value specified cannot exceed the logical record length minus the record key length. If the position is specified as zero, the record key does not appear in the data portion of unblocked records.

This information can be supplied by any of the three alternate sources.

NTM

specifies a number of tracks to be contained in a cylinder index before a higher level index is created. The maximum number is 99. (This operand is required only when a master index is specified in the OPTCD operand. Through this master index facility, extensive serial searches through a large index can be avoided).

If the cylinder index exceeds this number, a master index is created; if the master index exceeds this number, a higher level master index is created (up to a maximum of three levels of master index).

Based on the number of cylinders allocated to the data set, the control program calculates and establishes the required number of levels of master index. The calculations are based primarily on the following:

- The number of cylinders allocated to the data set.
- The KEYLEN value.
- The NTM value.

These calculations are performed when the data control block is opened. As the data set is loaded, entries are automatically made in the first level master index and, when appropriate, in the higher levels of master index.

This information can be specified by the DD statement or the user's problem program.

KEYLEN

specifies the length, in bytes, of the record key associated with a logical record. The maximum length of the record key is 255 bytes.

This information can be supplied by any of the three alternate sources.

CYLOFL

specifies the number of tracks to be reserved on each cylinder to hold records that overflow from other tracks on that cylinder. The maximum number of tracks that can be reserved is 99.

This information can be supplied by the DD statement or the user's problem program.

NOTE: Refer to Table 20 for a list of the situations in which the following four operands (BUFNO, BFALN, BUFL, and BUFCB) are applicable.

BUFNO

specifies the number of buffers to be assigned to the data control block. The maximum number that can be specified is 255; however, the number must not exceed the limit on input/output requests established during system generation.

This information can be supplied by the DD statement or the user's problem program.

BFALN

specifies the boundary alignment of each buffer, as follows:

F - the buffer starts on a full-word boundary (one that is not also a double-word boundary).
D - the buffer starts on a double-word boundary.

This information can be supplied by the DD statement or the user's problem program.

Table 20. QISAM Buffer Acquisition and Data Control Block Field Requirements

Usage and Characteristics		Method of Obtaining Buffer Pool		
		Automatically by OPEN	BUILD	GETPOOL
When Issued			In data control block exit routine or before OPEN	In data control block exit routine or before OPEN
Result		System acquires storage and structures into buffer pool	Structures storage into buffer pool	Acquires storage and structures into buffer pool
Data Control Block Requirements (to be provided no later than conclusion of data control block exit routine)	DCBBUFNO	Optional; if omitted, the field is not altered and two buffers are assumed	Required; user sets this field before or after BUILD is issued	Ignored; GETPOOL sets this field
	DCBBUFCB	Must be omitted; OPEN sets this field	Required; user sets this field before or after BUILD is issued	Ignored; GETPOOL sets this field
	DCBBFALN	Optional[1]	Ignored	Optional[1]
	DCBBUFL	Ignored; OPEN sets this field	Ignored[2]	Ignored[2]
Features		Provides standard options	More than one data control block can use pool	Execution time request for storage
Cautions		Only one data control block can use pool; must use FREEPOOL	User responsible for storage acquisition and buffer alignment; must close all data control blocks before reissuing BUILD	Only one data control block can use pool; must use FREEPOOL

[1]If omitted, field is not altered, double-word alignment is assumed.
[2]OPEN computes minimum buffer length and verifies that the buffer length specified in the length field of the buffer control block is at least as large as the computed length. The computed length is placed in the DCBBUFL field.

BUFL
specifies the length, in bytes, of each buffer to be obtained for a buffer pool. The maximum value is 32,760. Buffer requirements are given in "QISAM Load Mode Buffer Requirements" following the DCB macro-instruction description.

This information can be supplied by the DD statement or the user's problem program.

BUFCB
specifies the address of a buffer pool control block (i.e., the eight-byte field preceding the buffers in a buffer pool.)

The only alternate source for this information is the user's problem program.

EXLST
specifies the address of an exit list created by the programmer. The format of the list is presented in Appendix D.

Exit lists are required if data control block exit routines are used.

The only alternate source for this information is the user's problem program.

SYNAD
specifies the address of the user's synchronous error exit routine. The routine is entered if input/output errors result from an attempt to process data records.

The only alternate source for this information is the user's problem program.

CAUTIONS: The DCB macro-instruction must not be coded within the first 16 bytes of a control section. It can be preceded by padding, constants, or instructions.

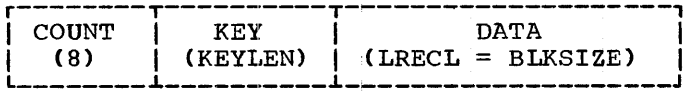
PROGRAMMING NOTES: After the data set is created, the following attributes are permanently established for the data set. When the indexed sequential method is used to gain access to the data set, it is controlled by these fixed characteristics.

Attributes

RECFM
LRECL
BLKSIZE
KEYLEN
RKP
CYLOFL
OPTCD
NTM

QISAM Load Mode Buffer Requirements

Fixed-length, unblocked records when the RKP field of the data control block equals zero:

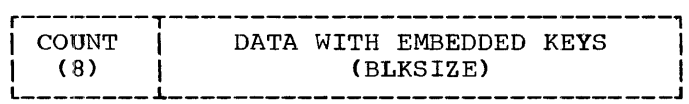


Buffer length is eight plus KEYLEN plus LRECL.

Fixed-length, unblocked records when the RKP field of the data control block is not equal to zero:

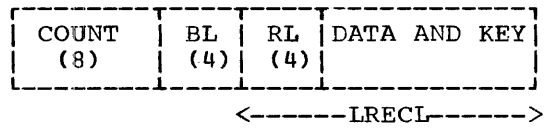
Buffer length is eight plus LRECL.

Fixed-length, blocked records:



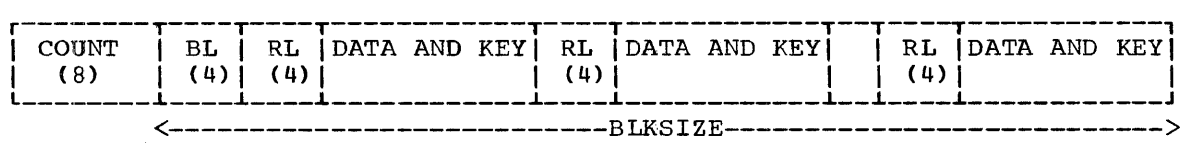
Buffer length is eight plus BLKSIZE.

Variable-length, unblocked records:



Buffer length is 12 plus LRECL.

Variable-length, blocked records:



Buffer length is eight plus BLKSIZE.

PUT -- Move Mode (R)

The PUT macro-instruction moves a logical record into an output buffer from which it is written.

Name	Operation	Operand	
[symbol]	PUT	{dcb-addrx (1)}	{area-addrx (0)}

dcb
: specifies the address of the data control block opened for the data set being created.

  If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

area
: specifies the address of the logical record to be moved into the buffer.

  If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

EXCEPTIONAL RETURNS: In both the load and scan modes, the user's synchronous error exit (SYNAD) routine is given control if any of the conditions listed in Table 21 arise. The control program notifies the user's problem program of each condition by setting the appropriate bit in the exceptional condition field of the data control block. The exceptional condition field is the two-byte area DCBEXCD1 and DCBEXCD2.

An all-zero exceptional field, or one in which only bit 3 of DCBEXCD2 is set to 1, indicates a successful completion. In either case, the SYNAD routine is not given control.

For all conditions, the general registers will contain the information listed in Table 22. The standard status information is listed in Appendix G.

Table 21. Contents of Exceptional Condition (DCBEXCD) Fields of Data Control Block -- QISAM Load and Scan Modes

Bit	Interpretation if Bit Set to 1	Set by
	(DCBEXCD1)	
0	Lower key limit not found	SETL (scan mode)
1	Invalid actual address for lower limit	SETL (scan mode)
2	Space not found in which to add a record	PUT (load mode)
3	Invalid request	SETL (scan mode)
4	Uncorrectable input error	GET (scan mode)
5	Uncorrectable output error	PUT or CLOSE (load mode) GET or CLOSE (scan mode)
6	Block could not be reached (input)	GET (scan mode)
7	Block could not be reached (output)	GET or CLOSE (scan mode)
	(DCBEXCD2)	
0	Sequence check	PUT (load mode)
1	Duplicate record	PUT (load mode)
2	Data control block closed when error routine entered	CLOSE
3	Overflow record	GET (scan mode)

Lower Key Limit Not Found: This condition is reported if the specified key or key class is not found in the data set.

Invalid Actual Address for Lower Limit: This condition is reported if the specified lower limit address is outside the space allocated to the data set.

Space Not Found in Which to Add a Record: This condition is reported if the space allocated to the data set is already filled. In the locate mode, a buffer segment address is not provided. In the move mode, data is not moved.

Invalid Request: This condition is reported if (1) the data set is already being sequentially referred to by the user's problem program, or (2) the buffer cannot contain the key and the data, or (3) the specified type is not also specified in the DCBOPTCD field of the data control block.

Uncorrectable Input Error: This condition is reported if the control program's error recovery procedures encounter an uncorrectable error in transferring a block from secondary storage to an input buffer. The

buffer address is placed in register 1, and the SYNAD routine is given control when a GET macro-instruction is issued for the first logical record.

Uncorrectable Output Error: This condition is reported if the control program's error recovery procedures encounter an uncorrectable error in transferring a block from an output buffer to secondary storage. If the error is encountered during closing of the data control block, bit 2 of DCBEXCD2 is set to 1 and the SYNAD routine is given control immediately. Otherwise, control program action depends on whether load mode or scan mode is being used.

If load mode is being used, the SYNAD routine is given control when the next PUT macro-instruction is issued. In the case of a failure to write a data block, register 1 contains the address of the output buffer; for other errors, register 1 contains all zeros. After appropriate analysis, the SYNAD routine should close the data set or end the job step. Subsequent execution of a PUT macro-instruction would cause reentry to the SYNAD routine, since an attempt to continue loading the data set would produce unpredictable results.

If scan mode is being used, the address of the output buffer is placed in register 1, and the SYNAD routine is given control when a GET macro-instruction is issued for the buffer. Buffer scheduling is suspended until the next GET macro-instruction is issued.

Block Could Not Be Reached (Input): This condition is reported if the control program's error recovery procedures encounter an uncorrectable error in searching on index or the track containing the block that is sought. The SYNAD routine is given control when a GET macro-instruction is issued for the first logical record of the unreachable block.

Block Could Not Be Reached (Output): This condition is reported if the control program's error recovery procedures encounter an uncorrectable error in searching an index or the track containing the block that is to be updated.

If the error is encountered during closing of the data control block, bit 2 of DCBEXCD2 is set to 1 and the SYNAD routine is given control immediately. Otherwise, the SYNAD routine is given control when a GET macro-instruction is issued for the first logical record of the unreachable block.

Sequence Check: This condition is reported if a PUT macro-instruction refers to a record whose key has a smaller numerical value than the key of the record previously referred to by PUT. The SYNAD routine is given control immediately; the record is not transferred to secondary storage.

Duplicate Record: This condition is reported if a PUT macro-instruction refers to a record whose key duplicates that of the record previously referred to by PUT. The SYNAD routine is given control immediately; the record is not transferred to secondary storage.

Data Control Block Closed When Error Routine Entered: This condition is reported if the control program's error recovery procedures encounter an uncorrectable output error during closing of the data control block. Bit 5 or bit 7 of DCBEXCD1 is set to 1, and the SYNAD routine is immediately given control. After appropriate analysis, the SYNAD routine must branch to the address in return register 14 so that the control program can finish closing the data control block.

Overflow Record: This condition is reported if the input record is an overflow record.

Table 22. Register Contents Upon Entry to SYNAD - QISAM Load and Scan Mode

Register	Contents
0	A pointer to status indicators (for GET or PUT macro-instruction) for uncorrectable input/output errors. In other cases, original contents destroyed.
1	The address of the buffer containing the record in error (for GET or PUT macro-instruction) for uncorrectable input/output errors. In other cases, original contents destroyed.
2 through 13	The contents that existed before the macro-instruction was executed.
14	The return address. This address is either (1) an address in the control program's CLOSE routine (when bit 2 of DCBEXCD2 is on), or (2) the address of the instruction following the expansion of the macro-instruction that caused the SYNAD routine to be given control (when bit 2 of DCBEXCD2 is off).
15	The address of the SYNAD routine.

EXAMPLE: In the following example, the data control block OUTDCB was opened for output. Successive PUT macro-instructions are issued to move records from a work area (WORKAREA) to output buffers.

```
        OPEN    (OUTDCB)
         .
         .
         .
EX1     PUT     OUTDCB,WORKAREA
         .
         .
         .
```

PROGRAMMING NOTES: After execution of the PUT macro-instruction, the device address of the block containing the logical record just processed is available in the eight-byte field DCBLPDA of the data control block. For blocked record formats, this address will be the same for each logical record within a block.

PUT -- Locate Mode (R)

The PUT macro-instruction supplies the address of the next available output buffer segment. The logical record can then be constructed in the buffer, by the programmer, for output as the next record in the data set being created. The address of the buffer is provided by the control program in register 1 after the PUT macro-instruction is executed.

Name	Operation	Operand
[symbol]	PUT	{dcb-addrx / (1)}

dcb
specifies the address of the data control block opened for the data set being created.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTIONS: The two modes of the PUT macro-instruction (locate and move) cannot be intermixed. The data set must be created by use of one or the other of the two modes.

When constructing blocked variable-length records in the locate mode, the user must choose one of the following methods of operations:

- Before executing each PUT macro-instruction, provide the actual logical record length in the DCBLRECL field of the data control block.
- Place a maximum logical record length, one that will not be altered, in the DCBLRECL field. This choice may result in more, but shorter, blocks, since the control program assumes that each logical record will require the maximum space when it tests to see if the next logical record can be contained in the current buffer.

EXCEPTIONAL RETURNS: Refer to the move-mode PUT macro-instruction.

PROGRAMMING NOTES: Refer to the move-mode PUT macro-instruction.

DCB -- Define Data Control Block for QISAM - Scan Mode

The DCB macro-instruction reserves space for a data control block (DCB) and informs the control program of the characteristics and intended uses of a data set.

Name	Operation	Operand
[symbol]	DCB	DSORG=IS,MACRF=code [,DDNAME=symbol][,BUFNO=absexp] [,BFALN={F|D}][,BUFL=absexp] [,BUFCB=relexp][,EODAD=relexp] [,EXLST=relexp][,SYNAD=relexp]

The keyword operands DSORG and MACRF can be supplied by only the DCB macro-instruction. The remaining operands can be supplied after assembly time by other sources; these sources are indicated in the operand descriptions.

DSORG
specifies the organization of the data set; IS specifies the indexed sequential organization.

MACRF
specifies the types of macro-instructions that will be used in processing the data set, as follows:

$$,MACRF=\left\{\begin{array}{l}(G\{M|L\})\\(G\{M|L\},S[K|I])\\(GL,PU)\\(GL,S[K|I],PU)\end{array}\right\}$$

G - GET macro-instruction (implies RELSE and ESETL macro-instruction)
  M - move-mode
  L - locate-mode
  S - SETL macro-instruction
  K - sequential processing using a record key or generic key
  I - sequential processing using a device address
P - PUTX macro-instruction
  U - indicates sequential updating

Omission of K or I implies sequential processing from the beginning of the data set.

DDNAME
specifies the name of the DD statement that will be used to describe the data set to be processed.

This information can also be supplied by the user's problem program before the data control block is opened.

NOTE: Refer to Table 20 for a list of situations in which the following four operands (BUFNO, BFALN, BUFL, and BUFCB) are applicable.

BUFNO
specifies the number of buffers to be assigned to the data control block. The maximum number that can be specified is 255; however, the number must not exceed the limit on input/output requests established during system generation.

This information can be supplied by the DD statement or the user's problem program.

BFALN
specifies the boundary alignment of each buffer, as follows:

F - the buffer starts on a full-word boundary (one that is not also a double-word boundary)
D - the buffer starts on a double-word boundary

This information can be supplied by the DD statement or the user's problem program.

BUFL
specifies the length, in bytes, of each buffer to be obtained for a buffer pool. The maximum value is 32,760. Buffer requirements are given in "QISAM Scan Mode Buffer Requirements" following the DCB macro-instruction description.

This information can be supplied by the DD statement or the user's problem program.

BUFCB
specifies the address of a buffer pool control block (i.e., the eight-byte field preceding the buffers in a buffer pool).

This information can be supplied by the user's problem program.

EXLST
specifies the address of an exit list created by the programmer. The format of the list is presented in Appendix D.

Exit lists are required if data control block exit routines are used.

The only alternate source for this information is the user's problem program.

EODAD
specifies the address of the user's end-of-data set exit routine for input data sets. This routine is entered when the user requests a record and there are no more blocks in the data set to be retrieved. If no routine has been provided, the task is abnormally terminated.

The only alternate source for this information is the user's problem program.

SYNAD
specifies the address of the user's synchronous error exit routine. The routine is entered as a result of errors detected during attempts to process records.

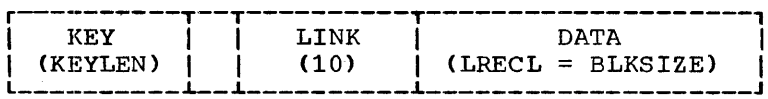
The only alternate source for this information is the user's problem program.

CAUTIONS: The DCB macro-instruction must not be written within the first 16 bytes of a control section. It can be preceded by padding, constants, or instructions.

PROGRAMMING NOTE: Refer to the load-mode DCB macro-instruction for those operands that are fixed when the data set is created.

QISAM Scan Mode Buffer Requirements

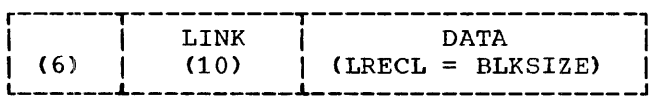
Fixed-length, unblocked records when both the key and data are to be read:



Buffer length is G plus LRECL, where G is the smallest multiple of eight equal to or greater than KEYLEN plus 10.

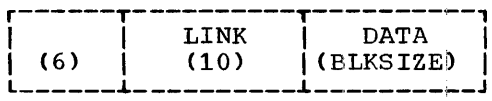
If the execution of the OPEN macro-instruction obtains buffers, the preceding buffer layout is used for fixed-length, unblocked records.

Fixed-length, unblocked records when only data is to be read:



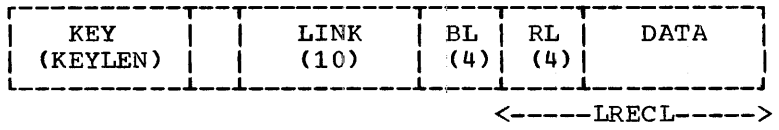
Buffer length is 16 plus LRECL.

Fixed-length, blocked records:



Buffer length is 16 plus BLKSIZE.

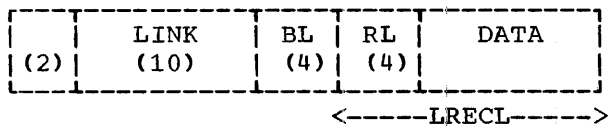
Variable-length, unblocked records when both the key and data are to be read:



Buffer length is H plus LRECL, where H is the smallest multiple of eight equal to or greater than KEYLEN plus 14.

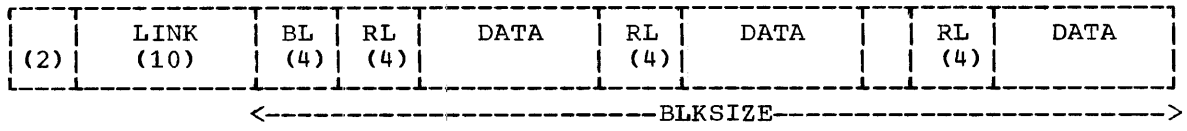
If the execution of the OPEN macro-instruction obtains buffers, the preceding buffer layout is used for variable-length, unblocked records.

Variable-length, unblocked records when only data is read:



Buffer length is 16 plus LRECL.

Variable-length, blocked records:



Buffer length is 12 plus BLKSIZE.

SETL -- Specify Start of Sequential Retrieval (R)

The SETL macro-instruction enables the user to retrieve records, starting at the beginning of an indexed sequential data set or at any point in the data set. Access other than at the beginning is specified by a record key, a generic key, or an actual address.

Name	Operation	Operand
[symbol]	SETL	{dcb-addrx / (1)},type-code[,{llimit-addrx / (0)}]

dcb
: specifies the address of the data control block opened for the data set being processed.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

type
specifies the starting point for processing, and any optional services requested, as follows:

Code	Starting Point
K	Record key
KC	Generic key
I	Actual address
B	Beginning of data set

Code	Service
D	Only the data portion of an unblocked record is to be retrieved. For blocked records this option is not meaningful and is ignored if specified.

The permissible combinations of these codes are listed in Table 23.

Table 23. Type Operand for SETL Macro-Instruction

Record Key	Generic Key	Actual Address	Beginning of Data Set
K	KC	I	B
KD	KCD	ID	BD

llimit
specifies the address of a field containing either the lower limit key or an actual address for sequential processing.

- If the type operand specified a record key or generic key, that key must be contained in the field.
- If the type operand specified an actual address, the field must contain an eight-byte actual address. Actual addresses are provided when the data set is created. Refer to the load mode PUT macro-instruction.
- If the type operand specified the start of the data set, the llimit operand should be omitted.

If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

EXCEPTIONAL RETURNS: Any errors resulting from the execution of this macro-instruction will cause control to be given to the synchronous error exit (SYNAD) routine specified in the data control block. The general register will be set as listed in Table 22 and the two-byte exceptional condition field in the data control block will be set as listed in Table 21.

EXAMPLE: In the following example, the data set associated with the INVEN data control block is to be scanned. The scan would start on a generic key supplied at the location RECKEY. Since D was written in the type operand, only the data portion of an unblocked record is read.

```
SETL INVEN,KCD,RECKEY
```

ESETL -- End Sequential Retrieval (R)

The ESETL macro-instruction ends the scanning of an indexed sequential data set.

Name	Operation	Operand
[symbol]	ESETL	{dcb-addrx / (1)}

dcb
specifies the address of the data control block opened for the data set being processed.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

GET -- Locate Mode (R)

The GET macro-instruction provides the address, in register 1, of the next logical record to be processed. If records are unblocked and record keys are also being read, register 0 will contain the address of the key.

Name	Operation	Operand
[symbol]	GET	{dcb-addrx / (1)}

dcb
specifies the address of the data control block opened for the data set being processed.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTIONS: If a GET macro-instruction is executed and the data set is not already in the scan mode, a scan mode will be initiated as if a SETL macro-instruction with a type operand of B (beginning of data set, read data plus key) had been executed.

The two modes of GET cannot be intermixed.

EXCEPTIONAL RETURNS: When the end of data set is reached, the EODAD routine specified in the data control block is given control.

Any error resulting from the execution of this macro-instruction will cause control to be passed to the user's synchronous error exit (SYNAD) routine. When this is done, the general registers will be set as listed in Table 22, and the two-byte exceptional condition field in the data control block will be set as listed in Table 21. The standard status indicators will be as listed in Appendix G. When bit 2 of DCBEXCD2 is off, register 14 may be adjusted to cause different actions upon return from the SYNAD routine. To accept the record, the user branches to the

address in register 14, which contains the address of the instruction after the GET macro-instruction expansion. To skip the record, the user adjusts the contents of register 14 to point to the beginning of the macro-expansion of the GET macro-instruction and then branches to the adjusted address in register 14. The user can also ignore the register 14 return address and either close the data control block or end the task.

PROGRAMMING NOTES: If the record provided was an overflow record, bit 3 in the DCBEXCD2 field of the data control block (Table 21) is set to 1. No exceptional return is made.

If the delete option is specified, logical records marked for deletion are not presented to the user for processing.

GET -- Move Mode (R)

The GET macro-instruction moves the next logical record to the user's work area.

Name	Operation	Operand
[symbol]	GET	{dcb-addrx / (1)},{area-addrx / (0)}

dcb
: specifies the address of the data control block opened for the data set being processed.

  If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

area
: specifies the address of the user's work area into which the control program moves the logical record. The work area requirements are given in "QISAM Scan Mode Work Area Requirements."

  If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

CAUTIONS: If a GET macro-instruction is executed and the data set is not already in the scan mode, a scan mode will be initiated as if a SETL macro-instruction with a type operand of B (beginning of data set, read data plus key) had been executed.

A move-mode GET macro-instruction may be used when records are to be retrieved from, but not returned to, a data set. If records are to be returned, then the locate-mode GET macro-instruction must be used. In any case, the two GET modes cannot be intermixed.

EXCEPTIONAL RETURNS: Refer to the locate-mode GET macro-instruction.

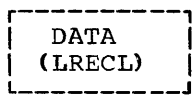
PROGRAMMING NOTES: Refer to the locate-mode GET macro-instruction.

QISAM Scan Mode Work Area Requirements

Fixed-length, unblocked records when both the key and data are to be read:

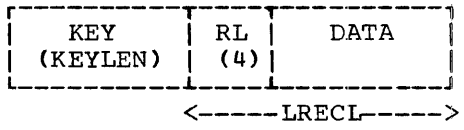
Work area length is KEYLEN plus LRECL.

Fixed-length, unblocked records when only data is to be read and fixed-length, blocked records:



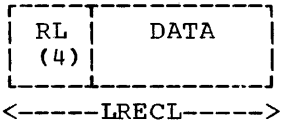
Work area length is LRECL.

Variable-length, unblocked records when both the key and data are to be read:



Work area length is KEYLEN plus LRECL.

Variable-length, unblocked records when only data is read and variable-length, blocked records:



Work area length is LRECL.

PUTX -- Update Mode (R)

The PUTX macro-instruction is used with the locate-mode GET macro-instruction when records are being updated within a buffer. It returns a record to the data set.

Name	Operation	Operand
[symbol]	PUTX	{dcb-addrx {(1)

dcb
specifies the address of the data control block opened for the data set being processed.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTIONS: The PUTX macro-instruction can only replace a record that was retrieved by a locate-mode GET macro-instruction.

For blocked records, if a PUTX macro-instruction was addressed to one of the logical records in a buffer, the contents of the entire buffer are written back to the data set after all the logical records have been processed.

To mark a logical record for deletion, the user must set the deletion-code byte to FF (hexadecimal) and then issue a PUTX macro-instruction for that record. In fixed-length-format records, the deletion-code is the first byte of the logical record; in variable-length-format records, it is the fifth byte of the logical record.

EXCEPTIONAL RETURNS: Any input/output error resulting from the execution of this macro-instruction will cause control to be passed to the user's synchronous error exit (SYNAD) routine. The general register will be set as listed in Table 22, and the exceptional condition field in the data control block will be set as listed in Table 21. The standard status indicators will be as listed in Appendix G.

EXAMPLE: In the following example, the OPEN macro-instruction opens a data control block STOCK. The SETL macro-instruction initiates a scan mode, using the generic key of the records placed at location PARTNO. The GET-PUTX processing loop updates the records, using the address provided in register 1 following the locate-mode GET macro-instruction. The ESETL macro-instruction would end the scan if the entire data set was not being processed.

```
AAX     OPEN   (STOCK)
          .
          .
          .
BAI     SETL   STOCK,KC,PARTNO
          .
          .
          .
BGS     GET    STOCK
          .
          .
          .
CTM     PUTX   STOCK
          .
          .
          .
        BH     CTQ
          .
          .
          .
        B      BGS
          .
          .
          .
CTQ     ESETL  STOCK
          .
          .
```

RELSE - Release Current Input Buffer (R)

The RELSE macro-instruction causes the remaining logical records of the input buffer to be ignored. The next GET macro-instruction will retrieve the first logical record from the next input buffer.

Name	Operation	Operand
[symbol]	RELSE	{dcb-addrx / (1)}

dcb
specifies the address of the data control block opened for the data set being processed.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTIONS: The RELSE macro-instruction can be used meaningfully with blocked records only. If used with unblocked records or the first logical record of a block, the request is ignored. If two consecutive release requests are issued, the second is ignored.

BASIC INDEXED SEQUENTIAL ACCESS METHOD (BISAM)

The basic indexed sequential access method (BISAM) provides direct storage and retrieval of the records in an indexed sequential data set. The BISAM macro-instructions permit direct:

- Retrieval of any logical record by its record key.
- Update-in-place of any logical record.
- Insertion of new logical records.

The dynamic buffer option provides the programmer with a special buffering facility for use with the basic indexed sequential access method.

The BISAM macro-instructions (DCB, READ, WRITE, and FREEDBUF) are used with the general service macro-instructions (BUILD, GETPOOL, FREEPOOL, GETBUF, FREEBUF, OPEN, and CLOSE). The opt_1 operand of the OPEN macro-instruction is ignored.

Macro-Instruction	Function
DCB	Constructs a data control block for an indexed sequential data set.
READ	Retrieves a logical record from an indexed sequential data set.
WRITE	Replaces an updated logical record, or adds a new logical record to an indexed sequential data set.
FREEDBUF	Releases a buffer obtained by use of the dynamic buffer options.

DCB -- Define Data Control Block for BISAM

This DCB macro-instruction reserves space for a data control block and informs the control program of the characteristics and intended uses of a data set.

Name	Operation	Operand
[symbol]	DCB	DSORG=IS,MACRF=code [,DDNAME=symbol][,NCP=absexp] [,MSHI=relexp,SMSI=absexp] [,MSWA=relexp,SMSW=absexp] [,BUFNO=absexp][,BFALN={F|D}] [,BUFL=absexp][,BUFCB=relexp] [,EXLST=relexp]

The keyword operands DSORG and MACRF can be supplied by only the DCB macro-instruction. The remaining operands can be supplied after assembly time by other sources; these sources are indicated in the operand descriptions.

DSORG
specifies the data set organization; IS specifies indexed sequential.

MACRF
specifies the types of macro-instructions that will be used in processing the data set, as follows:

```
          (R[S])
,MACRF=  {(W{U|A|UA})                }
          (R[S|U|US],W{U|A|UA})
```

R specifies READ macro-instruction (which can imply the FREEDBUF macro-instruction); optionally, the following can also be written:

S - provide dynamic buffering
U - read for update
US - read for update and provide dynamic buffering

W specifies WRITE macro-instruction; one of the following must be specified:

U - update records
A - add new records to the data set
UA - add new records and update existing records

DDNAME
specifies the name of the DD statement that will be used to describe the data set to be processed.

This information can also be supplied by the user's problem program before opening the data control block.

NCP
specifies the number of channel programs (lists of CCW's) to be established for this data control block. One channel program is

used for each outstanding READ or WRITE request (except WRITE type KN). If the number of outstanding input/output requests is greater than the number of channel programs available, the extra requests are queued until a channel program is made available by the completion of an input/output operation.

When the data set being processed resides on more than one direct-access device, the user's problem program should take advantage of this "overlap" potential and request a number of input/output operations.

The maximum number of channel programs is 99. (The user can have a greater number of outstanding read and write requests.)

This information can also be supplied by the DD statement or the user's problem program. If not supplied by any source, a value of 1 is assumed.

MSHI

specifies the address of a main storage area that the user has reserved for the highest level index. The control program will maintain the highest level index in that area until the data set is closed, thus permitting more efficient operation.

The only alternate source for this information is the user's problem program. If not provided by any source, the control program will search the highest level index on the direct-access device.

SMSI

specifies the number of bytes in the area reserved for the highest level index. When the data set was created, the system placed the minimum number of bytes required for the highest level index in the DCBNCRHI field of the data control block. The maximum value is 32,767.

The only alternate source for this information is the user's problem program. If the MSHI operand is omitted, this operand is ignored.

MSWA

specifies the address of a main storage work area reserved for the control program. This operand is required only when new variable-length records are being added to the data set.

If specified when fixed-length records are being added to the data set, the control program uses the work area to speed up record insertion.

The only alternate source for this information is the user's problem program.

SMSW

specifies the number of bytes reserved for the main storage work area. For unblocked records, the work area must be large enough to contain the count, key, and data fields of all the blocks on one track. For blocked records, the work area must be large enough to contain one logical record plus the count and data fields of all the blocks on one track. The maximum number is 32,767.

The only alternate source for this information is the user's problem program. If the MSWA operand is omitted, this operand is ignored.

NOTE: Refer to Table 24 for a list of the situations in which the following four operands (BUFNO, BFALN, BUFL, and BUFCB) are applicable.

BUFNO
specifies the number of buffers to be assigned to the data control block. The maximum number that can be specified is 255; however, the number must not exceed the limit on input/output requests established during system generation.

This information can also be supplied by the DD statement or the user's problem program.

BFALN
specifies the boundary alignment, in bytes, of each buffer, as follows:

F - the buffer starts on a full-word boundary (one that is not also a double-word boundary)
D - the buffer starts on a double-word boundary

This information can also be supplied by the DD statement or the user's problem program.

BUFL
specifies the length in bytes of each buffer to be obtained for a buffer pool. The maximum value is 32,760.

This information can also be supplied by the DD statement or the user's problem program.

BUFCB
specifies the address of a buffer pool control block (i.e., the eight-byte field preceding the buffers in a buffer pool).

The only alternate source for this information is the user's problem program.

EXLST
specifies the address of an exit list created by the programmer. The format of the list is presented in Appendix D.

Exit lists are required if the data control block exit is used.

The only alternate source for this information is the user's problem program before the data control block exit.

CAUTION: The DCB macro-instruction must not be written within the first 16 bytes of a control section. It can be preceded by padding, constants, or instructions.

PROGRAMMING NOTE: Refer to the programming notes of the load-mode DCB macro-instruction for those operands that are fixed when the data set is created.

Table 24. BISAM Buffer Acquisition and Data Control Block Field Requirements

Usage and Characteristics		Method of Obtaining Buffer Pool			User Performs All Buffering
		Dynamic Buffering	BUILD	GETPOOL	
When Issued			In data control block exit routine or before OPEN	In data control block exit routine or before OPEN	
Result		System acquires storage and structures into buffer pool	Structures storage into buffer pool	Acquires storage and structures into buffer pool	
Data Control Block Field Requirements (to be provided no later than conclusion of dcb exit routine)	DCBBUFNO	Optional[1]	Required; user sets this field before or after BUILD is issued	Ignored; GETPOOL sets this field	Ignored
	DCBBUFCB	Must be omitted; OPEN sets this field	Required; user sets this field before or after BUILD is issued	Ignored; GETPOOL sets this field	Must be omitted
	DCBBFALN	Optional[3]	Ignored	Optional[3]	Ignored
	DCBBUFL	Ignored;OPEN sets field	Ignored[2]	Ignored[2]	Ignored
Features		Control program does all management	More than one data control block can use pool; user issues GETBUF and FREEBUF	Execution time request for storage; user issues GETBUF and FREEBUF	User supplies buffers with own methods
Cautions		User responsible for freeing buffers through use of a WRITE or FREEDBUF	User responsible for storage acquisition & boundary alignment; must close all data control blocks before reissuing BUILD	Only one data control block can use pool; must use FREEPOOL	User responsible for boundary alignment

[1] If omitted, the field is not altered and two buffers are assumed.
[2] OPEN computes minimum buffer length and verifies that the buffer length specified in the length field of the buffer control block is at least as large as the computed length. The computed length is placed in the DCBBUFL field.
[3] If the field is omitted, double word alignment is assumed.

READ -- Retrieve a Logical Record (S)

The READ macro-instruction causes a logical record to be selected from an indexed sequential data set on the basis of a user-supplied record key. The block containing the selected logical record is placed in an area of main storage specified by the user. The address of the logical record is placed in the data event control block (DECB).

Name	Operation	Operand
[symbol]	READ	decb-symbol,type-{K|KU},dcb-addr ,area-{'S'|addr},length-{'S'|value} ,key-addr

decb
specifies the name to be assigned to the data event control block constructed as part of the expansion of the macro-instruction. The data event control block starts on a full-word boundary and contains:

- A parameter list.
- An event control block (ECB) that represents the event for which the problem program must wait before attempting to process the record.
- A pointer to the desired logical record.
- An exception code.

The format of the data event control block is shown in Table 25.

Table 25. Format of Data Event Control Block for BISAM

Offset from DECB Name (bytes)	Field
+0	Event control block
+4	Type
+6	Length
+8	Data control block address
+12	Area address
+16	Pointer to logical record
+20	Record key address
+24	Exception code (two bytes reserved; the second byte is used by the control program and must not be used by the problem program)

type
specifies the type of read operation with one of the following:

K - normal read
KU - read for update (the problem program will update the record and return it to the data set using a WRITE macro-instruction). In this type of operation, the control program remembers the device address of the record read for update; when the corresponding WRITE macro-instruction is executed, the index search that would otherwise be required is eliminated.

dcb
specifies the address of the data control block opened for the data set being processed.

area
specifies the address of the user's area into which the block is to be read. If 'S' is written, the control program will obtain an area of main storage for the block. The address of this area will be available in the data event control block after a WAIT macro-instruction has been issued for the read operation.

The area requirements are given in "BISAM Area Requirements."

length
specifies the number of bytes to be read. If this value is specified, it overrides the length known to the control program. An overriding length must not be specified for blocked records. If 'S' is written, the control program reads the entire block.

key
specifies the address of a field containing the record key for the desired logical record.

CAUTIONS: The READ macro-instruction may return control before the actual transmission of data begins or is completed.

To determine whether the read operation has been completed, the WAIT macro-instruction must be issued before an attempt is made to use the data transferred into the area designated by the area operand. The data event control block used for a read operation should not be reused until the WAIT macro-instruction has been issued. After the wait has been satisfied, the first byte of the exception code of the data event control block must be examined to ensure that the operation was successfully completed.

EXCEPTIONAL RETURNS: A one-byte exceptional condition code is set in the data event control block if the read operation could not be completed without some error. The possible read errors, and their corresponding bit settings, are listed in Table 26.

Table 26. Contents of Exceptional Condition Code Byte, Data Event Control Block - BISAM

Bit	Interpretation if Bit Is Set to 1	Set By		
		READ	WRITE (update)	WRITE (add)
0	Record not found	X	X	
1	Record length check	X	X	X
2	Space not found in which to add a record			X
3	Invalid request		X	
4	Uncorrectable input/output error	X	X	X
5	Unreachable block	X	X	X
6	Overflow record	X		
7	Duplicate record presented for inclusion in the data set			X

An all-zero exception code or one in which only bit 6 is set to 1 indicates successful completion.

Record Not Found: This condition is reported if the logical record with the specified key is not found in the data set.

Record Length Check: This condition is reported, for READ and update WRITE macro-instructions, if an overriding length is specified and (1) the record format is blocked or (2) the record format is unblocked but the overriding length is greater than the length known to the control program. This condition is reported, for the add WRITE macro-instruction, if an overriding length is specified.

When blocked records are being updated, the control program must find the high key in the block in order to write the block. (The high key is not necessarily the same as the key supplied by the problem program.) The high key is needed for writing because the control unit for direct-access devices permits writing only if a search on equal is satisfied; this search can be satisfied only with the high key in the block. If the user were permitted to specify an overriding length shorter than the block length, the high key might not be read; then, a subsequent write request could not be satisfied. In addition, failure to write a high key during update would make a subsequent update impossible.

Space Not Found in Which to Add a Record: This condition is reported if no room exists in either the appropriate cylinder overflow area or the independent overflow area when a new record is to be added to the data set. The overflowing record may be the new record or a record pushed out of the prime data track by the new record.

If the pointer-to-logical-record field in the data extent control block (DECB) is zero, the location of the overflowing record is in the area-address and record-key-address fields of the DECB. If the pointer-to-logical-record field is not zero, this field itself contains the location of the overflowing record; this location contains the key followed by a 10-byte link field, followed by data.

Invalid Request: This condition is reported if byte 25 of the data event control block indicates that this request is an update WRITE macro-instruction corresponding to a READ for update macro-instruction, but the input/output block (IOB) for the READ is not found in the update queue. This condition could be caused by the problem program altering the contents of byte 25 of the data extent control block.

Uncorrectable Input/Output Error: This condition is reported if the control program's error recovery procedures encounter an uncorrectable error in transferring data between main and secondary storage.

Unreachable Block: This condition is reported if the control program's error recovery procedures encounter an uncorrectable error while performing a function other than transferring data between main and secondary storage. Such an error could occur in searching an index or in following an overflow chain.

Overflow Record: This condition is reported if the record just read is an overflow record.

Duplicate Record Presented for Inclusion in the Data Set: This condition is reported if the new record to be added has the same key as a record in the data set. However, if the delete option was specified and the record in the data set is marked for deletion, this condition is not reported. Instead the new record replaces the existing record.

If the record format is blocked and the relative key position is zero, the new record cannot replace an existing record that is of equal Key and is marked for deletion.

NOTE: If a READ macro-instruction for which the control program obtains an area ends in error, the problem program must free the area using a FREEDBUF macro-instruction.

EXAMPLE: In the following example, the block containing the logical record whose record key is found at location RECKEY will be read into a main storage area provided by the control program. The address of the logical record is placed in the data event control block DECBI. The data set is described by the data control block INPUT.

```
READ   DECBI,K,INPUT,'S',500,RECKEY
```

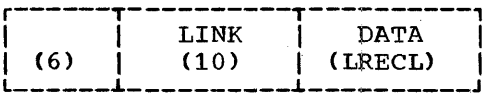
L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B, except for the following special operand requirements:

Operand	L Form	E Form
decb	required	required
type	required	required
MF	required	the operand must be written as MF=E

The operand MF=E does not require a parameter list address because the first operand, decb, is used as a pointer to a parameter list that was established by the L form of the macro-instruction.

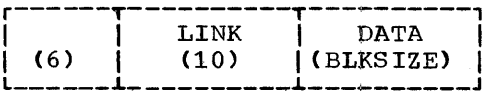
BISAM Area Requirements

Fixed-length, unblocked records processed by READ and update WRITE macro-instructions or fixed-length, blocked or unblocked records processed by add WRITE macro-instructions:



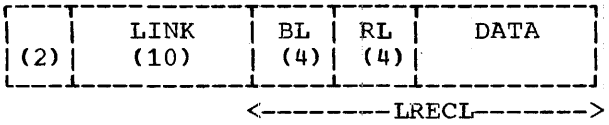
Area length is 16 plus LRECL.

Fixed-length, blocked records processed by READ and update WRITE macro-instructions:



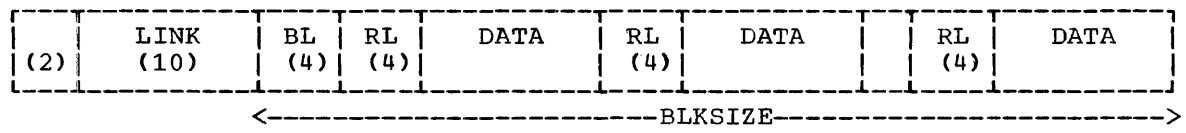
Area length is 16 plus BLKSIZE.

Variable-length, unblocked records processed by READ and update WRITE macro-instructions or variable-length, blocked or unblocked records processed by add WRITE macro-instructions:



Area length is 16 plus LRECL.

Variable-length, blocked records processed by READ and update WRITE macro-instructions:



WRITE -- Write a Logical Record (S)

The WRITE macro-instruction is used either to replace blocks that have been updated or to add new logical records to an already existing indexed sequential data set. If the data set consists of blocked records, the block containing the updated logical record is written into the data set.

Name	Operation	Operand
[symbol]	WRITE	decb-symbol,type-{K|KN},dcb-addr ,area-{'S'|addr},length-{'S'|value} ,key-addr

decb
: specifies the name to be assigned to the data event control block (DECB) constructed as part of the expansion of the macro-instruction. The format and use of the data event control block are as explained in the READ macro-instruction.

type
: specifies one of the following as the type of write operation:

  K - write a block containing an updated logical record (one that was retrieved by the READ macro-instruction) or replace an unblocked record without having read that record.
  KN - write a new logical record in the data set

dcb
: specifies the address of the data control block opened for the data set being processed.

area
: specifies the address from which the record is to be written. If 'S' was specified in the area operand of the corresponding READ macro-instruction, it can be specified here. 'S' indicates that the programmer has no need for the main storage once the block is written. In this case, the area field in the data event control blocks for the READ and WRITE macro-instructions must contain the same addresses. (After the WRITE macro-instruction has been executed, the control program will reuse or release the main storage formerly occupied by the block.) If new records are being written, 'S' can not be specified in the area operand.

  The area requirements are given in "BISAM Area Requirements."

length
specifies the number of bytes to be written. The value overrides the length known to the control program. When new records are being added to a data set or when blocked records are being updated, an overriding length may not be specified. If 'S' is written, the length known to the control program is used.

key
specifies the address of a field containing the record key of a logical record. When blocked records are being updated, this is the record key of the updated logical record which is not necessarily the high key in the block.

CAUTIONS: When blocked records are being updated, the overriding length should be used with extreme caution, both in the WRITE macro-instruction and in the corresponding READ macro-instruction. In order to write back the block, the control program must find the high key (which is not necessarily the same as the key supplied by the user) in the block. The control unit for direct-access devices permits writing only if a search on "equal" is satisfied; this search can be satisfied only with the high key in the block. If the user specifies an overriding length shorter than the block length, the high key might not be read and the subsequent write request could not be satisfied. Also, the failure to write out the high key would make a subsequent update impossible.

EXCEPTIONAL RETURNS: A one-byte exceptional condition code is set in the data event control block if the write operation could not be completed without some error. The possible write errors, and their corresponding bit settings, are listed in Table 26.

L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B except for the following special operand requirements:

Operand	L Form	E Form
decb	required	required
type	required	required
MF	required	the operand must be written as MF=E

The operand MF=E does not require a parameter list address because the first operand, decb, is used as a pointer to a parameter list that was established by the L form of the macro-instruction.

FREEDBUF -- Free Dynamically Obtained Buffer (R)

The FREEDBUF macro-instruction releases a buffer originally obtained by the control program in response to an 'S' in the area operand of a READ macro-instruction.

Name	Operation	Operand
[symbol]	FREEDBUF	{decb-addrx / (0)},type-K,{dcb-addrx / (1)}

decb
specifies the address of a data event control block (DECB) used in a READ macro-instruction that specified an 'S' in the area operand.

If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

type
specified as K to indicate the indexed sequential access method.

dcb
specifies the address of the data control block opened for the data set being processed.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTIONS: The area field in the data event control block must not be altered between execution of the READ and FREEDBUF macro-instructions.

The buffer can also be released by a WRITE macro-instruction.

BASIC DIRECT ACCESS METHOD (BDAM)

The macro-instructions for the basic direct-access method (BDAM) are provided specifically for use with direct-access devices. The programmer can perform input/output operations directly by specifying any of the following:

- An actual address -- the location of a block on a direct access device.
- A relative block address -- the relative position of a block within a data set.
- A relative track address -- the relative track within a data set at which the control program is to begin a search

A special buffering facility (the dynamic buffer option) enables the programmer to request control program buffer management.

The BDAM macro-instructions are DCB, READ, WRITE, FREEDBUF, and RELEX. The general service macro-instructions (BUILD, GETPOOL, FREEPOOL, GETBUF, FREEBUF, OPEN, and CLOSE), can also be used with the basic direct access method.

Macro-Instruction	Function
DCB	Defines a data control block for BDAM
READ	Retrieves a block
WRITE	Writes a block
RELEX	Releases exclusive control of a record
FREEDBUF	Frees a buffer obtained by the dynamic buffer option

The OPEN macro-instruction option specifying the intended method of input/output processing, opt_1, has the following effect on BDAM macro-instructions:

OPEN Macro-Instruction Operands	Effect
INPUT	Any write will be handled as an invalid request.
OUTPUT	All read or write requests are processed.
UPDAT	All read or write requests are processed.

DCB -- Define Data Control Block for BDAM

The DCB macro-instruction reserves space for a data control block and informs the control program of the characteristics and intended uses of a data set.

Name	Operation	Operand
[symbol]	DCB	DSORG={DA|DAU},MACRF=code [,DDNAME=symbol][,OPTCD=code] [,RECFM={U|V|F|FT}][,KEYLEN=absexp] [,LIMCT=absexp][,BLKSIZE=absexp] [,BUFNO=absexp][,BFALN={F|D}] [,BUFL=absexp][,BUFCB=relexp] [,EXLST=relexp]

The keyword operands DSORG and MACRF can be specified in only the DCB macro-instruction. The remaining operands can be supplied after assembly by other sources; these sources are indicated in the operand descriptions.

DSORG
specifies the organization of the data set as one of the following:

DA - direct organization
DAU - direct organization in which any data set contains location-dependent information with respect to this data set.

MACRF
specifies the types of macro-instructions that will be used in processing the data set, as follows:

```
                 (R{K|I|KI}[X|S|XS])
,MACRF=          (W{A|K|I|AK|AKI|KI|AI})
                 (R{K|I|KI}[X|S|XS],W{A|K|I|AK|AKI|KI|AI})
```

Code	Modifier	Function Specified
R		READ macro-instruction
	K	Search argument is a key
	I	Search argument is a block identification
	X	Retain exclusive control of a block
	S	Dynamic buffering
W		WRITE macro-instruction
	A	Blocks are to be added to the data set
	K	Search argument is a key
	I	Search argument is a block identification

DDNAME
specifies the name of the DD statement that will be used to describe the data set to be processed.

This information can also be supplied by the user's problem program before opening the data control block.

OPTCD
specifies an optional service to be provided by the control program, as follows:

W - write validity check
F - feedback will be requested in READ or WRITE macro-instructions in the program; it is to be in the form specified by the code R or A, below, or a default form if both R and A are omitted
E - extended search
R - relative block addresses are to be used
A - actual addresses are to be used; the extended search option is ignored, if specified

If neither R nor A is specified, relative track addressing is assumed.

This information can be supplied by the DD statement or the user's problem program.

RECFM
specifies the record format. One of the following is written:

U - undefined records
V - variable-length records
F - fixed-length records
FT - fixed-length records; track overflow is used

This information can be supplied by any of the three alternate sources.

If no record format information is supplied, a format-U record is assumed.

KEYLEN
specifies the length, in bytes, of the key for each physical record. The maximum length is 255 bytes.

This information can be supplied by any of the alternate sources. If not supplied by any source, a read or write request requiring a key is treated as invalid.

LIMCT
specifies the maximum number of blocks or tracks to be searched when the extended search option has been chosen. The number of blocks is specified when relative block addresses are used; the number of tracks is specified when relative track addresses are used. If the value equals or exceeds the number of tracks or blocks in the data set, the entire data set will be searched until the block is found.

This information can be supplied by the DD statement or the user's problem program. If the value is not supplied by any source, a request for an extended search is invalid.

BLKSIZE
specifies the maximum block length. The maximum number is 32,760.

This information can be supplied by any of the three alternate sources.

NOTE: Refer to Table 27 for a list of the situations in which the following four operands (BUFNO, BUFALN, BUFL, and BUFCB) are applicable.

BUFNO
specifies the number of buffers to be maintained by the control program when the dynamic buffering option is specified in the MACRF operand. The maximum number that can be specified is 255; however, the number must not exceed the limit on input/output requests established during system generation.

This information can also be supplied by the DD statement or the user's problem program. If the number is not supplied by any source and the dynamic buffering option is requested in the MACRF operand, a value of 2 is assumed.

Note: Dynamic buffers are allocated only when 'S' is specified in the area operand of a READ macro-instruction.

BFALN
specifies the boundary alignment of each buffer, as follows:

F - the buffer starts on a full-word boundary (one that is not also a double-word boundary)
D - the buffer starts on a double-word boundary

This information can also be supplied by the DD statement or the user's problem program.

BUFL
specifies the length, in bytes, of each buffer provided by the control program. If dynamic buffering is used and the key operand of the READ or WRITE macro-instruction is 'S', this length must be that of the key and data. The maximum number is 32,760.

This information can be supplied by any of the alternate sources.

BUFCB
specifies the address of a buffer pool control block (i.e., the eight-byte field preceding the buffers in a pool).

The only alternate source for this information is the user's problem program.

EXLST
specifies the address of an exit list created by the programmer. The format of the exit list is shown in Appendix D.

Exit lists are required if data control block exit routines are used.

The only alternate source for this information is the user's problem program.

CAUTION: The DCB macro-instruction must not be written within the first 16 bytes of a control section. It can be preceded by padding, constants, or coding.

Table 27. BDAM Buffer Acquisition and Data Control Block Field Requirements

Usage and Characteristics		Method of Obtaining Buffer Pool: Dynamic Buffering	Method of Obtaining Buffer Pool: BUILD	Method of Obtaining Buffer Pool: GETPOOL	User Performs All Buffering
When Issued			In data control block exit routine or before OPEN	In data control block exit routine or before OPEN	
Result		System acquires storage and structures into buffer pool	Structures storage into buffer pool	Acquires storage and structures into buffer pool	
Data Control Block Field Requirements (to be provided no later than conclusion of data control block exit routine)	DCBBUFNO	Optional; if omitted, the field is not altered and two buffers are assumed	Required; user sets this field before or after BUILD is issued	Ignored; GETPOOL sets this field	Ignored
	DCBBUFCB	Ignored; OPEN sets this field	Required; user sets this field before or after BUILD is issued	Ignored; GETPOOL sets this field	Ignored
	DCBBFALN	Optional[1]	Ignored	Optional[1]	Ignored
	DCBBUFL	Required	Ignored	Ignored	Ignored
Features		Control program does all management	More than 1 data control block can use pool; user issues GETBUF and FREEBUF	Execution time request for storage; user issues GETBUF and FREEBUF	User supplies buffers through his own methods
Cautions		User responsible for freeing buffers through use of a WRITE or FREEDBUF	User responsible for storage acquisition and boundary alignment; must close all data control blocks before reissuing BUILD	Only one data control block can use pool; must use FREEPOOL	User responsible for boundary alignment

[1]If omitted, field is not altered, double-word alignment is assumed.

READ -- Read a Block (S)

The READ macro-instruction retrieves a block from a data set. To request a block, one of the following can be specified:

- The relative block address.
- A relative track plus either a block identification or key.
- The actual address of the block.

Name	Operation	Operand
[symbol]	READ	decb-symbol,type-code,dcb-addr ,area-{'S'|addr},length-{'S'|value} ,key-{'S'|addr},blkref-addr

decb
specifies the name to be assigned to the data event control block (DECB) constructed as part of the expansion of the macro-instruction. The data event control block starts on a full-word boundary and contains the following:

- parameter list
- event control block (ECB) that is tested for completion of the read operation
- exception code

The format of the data event control block is shown in Table 28.

Table 28. Data Event Control Block for BDAM

Offset from DECB name (bytes)	Field
+0	Event control block
+1	Two-byte exception code following input/output operation
+4	Type
+6	Length
+8	Data control block address
+12	Area address
+16	Block reference address
+20	Key address

type
specifies the type of read operation, as follows:

D - direct-access method
I - locate the block using a block identification
K - locate the block using a key
F - provide position feedback (as an actual address unless a different form is specified in the DCBOPCD field.)
X - maintain exclusive control of the block and provide position feedback

The above values determine the action of the control program as follows:

- If I is written, the data and key (if keys are used) portion of the block is read. The search for the block is confined to the track where the search begins. If the block is not found, bit 8 in the exception code is set to 1.

- If K is written, only the data is read. Unless the extended search option is specified, the search for the block is confined to the track where the search begins. If the block is not found, bit 8 in the exception code is set to 1.

- If F is written, position feedback is provided in the field specified by the blkref operand. Feedback is provided in the form specified in the DCBOPTCD field of the data control block. If no form is specified in the data control block, the feedback is in the form of an actual address.

- If X is written, no other program using the same data control block, and requesting exclusive control, can read the block until the first requestor releases the block. Feedback is provided as if F were written.

The combinations in which the type operand can be specified are summarized in Table 29.

Table 29. READ Macro-Instruction Type Operand Values for BDAM

Type Operand	Interpretation
DI	Search uses block identification
DK	Search uses key
DIF	Search uses block identification; feedback requested
DIX	Search uses block identification; maintain exclusive control; feedback implied
DIFX	Equivalent to DIX
DKF	Search uses key; feedback requested
DKX	Search uses key; maintain exclusive control; feedback implied
DKFX	Equivalent to DKX

dcb
: specifies the address of the data control block opened for the data set being processed.

area
: specifies the address of the user's main storage area that is to contain the data portion of the block. If S was written in the MACRF operand of the DCB macro-instruction, 'S' can be coded for this operand. In this case, the control program obtains an input area and places its address in the area address field of the data event control block. The address is available in the data event control block after completion of the input/output operation.

  An area provided by the control program must be returned by a WRITE or FREEDBUF macro-instruction.

length
specifies the number of data bytes to be read. The maximum number is 32,760. If 'S' is written, the control program determines the block length from the DCBBLKSI field in the data control block.

key
specifies the address of an area in main storage. The use of this area depends on how the type operand was specified, as follows:

- If the search uses a key, this operand specifies the address of the field containing the key.
- If the search uses a block identification, this operand specifies the address of a field into which the control program is to read the key of the block when retrieved. The key is available after completion of the input/output operation. If the key operand is given a value of zero, the control program will not read the key into main storage.

If the area operand was written as 'S', this operand can also be written as 'S'. The 'S' is interpreted as meaning that when the search uses a block identification, the control program is to provide the input area, and the key and data are to be read sequentially into that area.

This operand is effective only if the DCBKEYLE field is non-zero.

blkref
specifies the address of the field containing the reference to be used in retrieving the block. The reference can be by relative block address, by relative track (with or without block position on that track), or by actual address. Relative block references are permitted only for format-F records without track overflow.

The field containing the block reference is also the field in which position feedback is provided when F or X is written in the type operand. If the initial reference and the feedback are both relative addresses, the blkref field need be only three bytes in length; if either is an actual address, the field must be eight bytes in length. If the field is eight bytes in length, feedback in the form of a relative address will be right-justified in the three leftmost bytes of the field, and a reference in the form of a relative address must also be right-justified in the three leftmost bytes of the field.

CAUTIONS: The READ macro-instruction may return control before the actual transmission of data begins or is completed.

To determine whether or not the read operation has been completed, the WAIT macro-instruction must be issued before an attempt is made to use the data transferred into main storage. The data event control block employed for a read operation should not be reused until the WAIT macro-instruction has been issued.

If the user requests dynamic buffering and issues more read requests than there are buffers available to hold the input blocks, the extra requests are queued. The program will wait permanently if there are n buffers available and the user waits on the n+1 request.

EXCEPTIONAL RETURNS: A two-byte exceptional condition code is set in the data event control block if the read operation could not be completed correctly. The possible errors, and their corresponding bit settings, are listed in Table 30.

Note: DECB bits 2 through 7 of byte 0 are always zero. Byte 1 is set to reflect error conditions. Byte 2 is meaningful only when bit 11 is set to one.

Table 30. Exception Condition Bits for BDAM

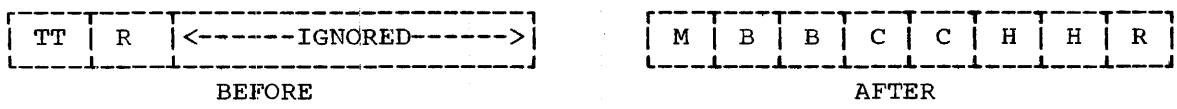
ECB Bit	Condition Present if Set to 1	Set ByCB	
	Byte 1	READ	WRITE
8	Block not found within search limits.	X	X
9	Length specified differs from actual length.	X	X
10	Space to add a block not available within search limits.		X
11	Invalid request. (See byte 2.)	X	X
12	Uncorrectable input/output error.	X	X
13	End of data (block of zero data length).	X	X
14	Unrelated error that could not be corrected.	X	X
15	Updated block could not be found on exclusive list (WRITE with type X only).		X
	Byte 2		
16	Not used		
17	Write request for data control block opened as input.		X
18	Request specified extended search but field DCBLIMCT contained zero.	X	X
19	Specified block reference not within the limit of tracks or blocks allotted to this data set.	X	X
20	Protection violation for capacity record.		X
21	Specified a key and block reference when the data control block indicated that no keys were used, or no key address has been supplied.	X	X
22	Exercised options not called for by data control block.	X	X
23	Attempted to add a new block whose key began with FF (hexadecimal) byte (fixed-length records only).		X

EXAMPLE: The following example shows the use of the READ macro-instruction when a relative track address is supplied. A block from the data set associated with the REPORT data control block will be read into an area provided by the system. The length read will be the maximum length as stated in the data control block. The block reference is located at the field RECADD. The type operand specifies that the search is to be on a block identification, and that exclusive control of the block is requested.

```
        OPEN    (REPORT,UPDAT)
        .
        .
        .
EX1     READ    INDECB,DIX,REPORT,'S','S',KEYADD,RECADD
        .
        .
        .
        WAIT    ECB=INDECB
```

Automatic feedback will be provided, because X was written. Assuming that the data control block did not specify feedback, the field referred to by the blkref operand (RECADD) is assumed to be eight bytes. The programmer's reference is initially provided in the three high-order bytes.

Feedback is in the form:



Assuming that DCBKEYLE is not zero, the control program will place the key in the field KEYADD. The user then waits for the completion of the read operation, using the name of the data event control block INDECB.

L- AND E-FORM USE: The L and E forms of this macro-instruction are written as described in Appendix B except for the following special operand requirements:

Operand	L Form	E Form
decb	required	required
type	required	required
MF	required	the operand must be written as MF=E

The operand MF=E does not require a parameter list address because the first operand, decb, is used as a pointer to a parameter list that was established by the L form of the macro-instruction.

WRITE -- Write a Block (S)

The WRITE macro-instruction replaces blocks and adds new blocks to an existing direct-access data set.

Name	Operation	Operand
symbol	WRITE	decb-symbol,type-code,dcb-addr ,area-{'S'|addr} ,length-{'S'|value},key-{'S'|addr} ,blkref-addr

decb

specifies the name to be assigned to the data event control block (DECB) constructed as part of the expansion of the macro-instruction. The format of the data event control block and its contents are explained in the READ macro-instruction.

type

specifies the type of write operation, as follows;

D - direct-access method
I - search argument is a block identification
K - search argument is a key
A - a new block is to be added to the data set
F - requests device address feedback
X - write a block that had been read with the exclusive option

The type values determine the action of the control program as follows:

- When I is specified, the search for the block is confined to the track where the search begins. If the block is not found, bit 8 in the exception code is set 1. The key (if keys are used) and data portions of the block are written.

- When K is specified, the search for the block is confined to the track where the search begins unless the extended search option was chosen. If the block is not found, bit 8 in the exception code is set to 1. Only data is written.

- When A is specified, the search for space is confined to the track where the search begins unless the extended search option was chosen. If space is not found, bit 10 in the exception code is set to 1. The key (if keys are used) and data portions of the block are written.

The combinations in which the type operand can be specified are listed in Table 31.

The form of feedback provided depends on whether the data control block specifies the feedback option.

dcb

specifies the address of the data control block opened for the data set being processed.

area

specifies the address of the user's main storage area containing the output block. If the area was originally obtained by the control program (through an 'S' in the area operand of the READ macro-instruction), the user can release the area by specifying 'S' in this macro-instruction.

Before the write request is executed, the area address must be moved from the input data event control block to the output data event control block.

Table 31. WRITE Macro-Instruction Type Operand Values for BDAM

Code	Interpretation
DI	Search uses block identification; any attempt to write a capacity record (record zero) will be treated as an invalid request.
DIF	Search uses block identification[1]
DIX	Search uses block identification; release exclusive control[1]
DIFX	Equivalent to DIX.
DK	Search uses a key.
DKF	Search uses key; feedback requested.
DKX	Search uses key; release exclusive control; no feedback.
DKFX	Equivalent to DKX.
DA	Add a new block wherever there is space.
DAF	Add a new block whenever there is space; feedback requested.
[1]This indicates that the field specified by the blkref operand contains feedback from a previous read operation.	

length
specifies a value for the number of data bytes (excluding key bytes) to be written for format F or U blocks. For format V blocks, the system places the block length supplied in the data block into the DECB.

If the number of bytes to be written is less than that portion of the block being replaced, the remainder of the block is padded with binary zeros. If the number of bytes to be written is greater than the portion of the block being replaced, the writing stops when the end of the block is reached.

The maximum number is 32,760. A length specification of zero indicates that an end-of-data mark or indicator is to be written. (Valid only for a type DA or DAF request.)

If 'S' is written, the control program will write the maximum length block stated in the data control block for format F or U blocks.

key
specifies the address of a field containing the key of the block to be written. The key is used as a search argument if the type operand was specified as DK, DKX or DKF. If the type operand was written as DI, DIF or DIX, the key is written into the data set when the block is written. If a key is not to be written, the key operand must have a value of zero.

To release a key field obtained by the control program in response to a read request, 'S' must be written in the key operand. The address of the key field must be moved from the input DECB to the output DECB before the WRITE macro-instruction is executed. This operand is effective only if the DCBKEYLE field in non-zero.

blkref

specifies the address of the field containing the address reference to be used in writing the block. The reference can be by relative block address, by relative track (with or without block position on that track), or by actual address. Relative block references are permitted only for format-F records without track overflow.

When writing by block identification with exclusive control (types DIX and DIFX), the reference in the blkref field is assumed to be an actual address unless the data control block specifies feedback to be a relative address. The contents of the blkref field should not be changed between execution of READ and WRITE macro-instructions.

The length of the blkref field depends on the type of reference provided and whether the feedback option is specified in the data control block or the WRITE macro-instruction, or both. (Refer to the READ macro-instruction for details.)

CAUTIONS: The WRITE macro-instruction may return control before the actual transmission of data begins or is completed.

To determine whether or not the write operation has been completed, a WAIT macro-instruction must be issued before the area containing the output block is reused.

The data event control block employed for a write operation should not be reused until the WAIT macro-instruction has been issued.

EXCEPTIONAL RETURNS: A two-byte exceptional condition code is set in the data event control block if the write operation could not be completed successfully. The possible errors, and their corresponding bit settings, are listed in Table 30.

EXAMPLES: In the following examples, EX1 shows the use of a WRITE macro-instruction when keys are supplied as search arguments. Assume that the data control block INVEN specifies the feedback option; 800 data bytes will be written from the field WORK using the search argument found at KEYADD. The search starts at the relative track specified by the TRACKREF field.

The true relative track will be returned as feedback in the TRACKREF field when the write operation is completed (if feedback option was specified in the data control block; otherwise the actual address will be returned).

```
EX1     WRITE   OUTDECB,DKF,INVEN,WORK,800,KEYADD,TRACKREF
```

In the second example, the READ macro-instruction requests that a block from the data set ORDERS be retrieved and placed in an area provided by the control program. The search is to use a key located in the field NEWKEY and a relative track number found in the field FIND. Assume that feedback was requested in the data control block. Since feedback is requested by an F in the type operand, the control program returns a three-byte relative address in the field FIND when the block is read.

After the user waits for the read operation to be completed, the address of the dynamic buffer area is found in the data event control block DECB5. The user can either update the block or ignore it and read another block. If the block is not updated, the FREEDBUF macro-instruction is used to return the buffer to the pool. If the block is processed, the WRITE macro-instruction is used to return it to the data

set, and the buffer to the pool. The key is not required, since an exact relative address has been provided; the key operand has a value of zero and the key will not be written. An 'S' is specified in the area operand to release the dynamic buffer area.

```
        .
        .
        .
ETC     READ      DECB5,DKF,ORDERS,'S','S',NEWKEY,FIND
        .
        .
        .
        WAIT      DECB5
        .
        .
        .
        BH        LETGO (test for update)
        .
        .
        .
        WRITE     DECB6,DIF,ORDERS,'S','S',0,FIND
        .
        .
        .
        WAIT      DECB6
        .
        .
        .
        B         ETC
LETGO   FREEDBUF  DECB5,D,ORDERS
        B         ETC
        .
        .
        .
FIND    DS        FL3
```

L- AND E-FORM USE: Refer to the READ macro-instruction.

RELEX -- Release Exclusive Control (R)

The RELEX macro-instruction releases from exclusive status a block that was requested in a READ (type DX) macro-instruction. This permits other exclusive requests to gain access to the block.

Name	Operation	Operand
[symbol]	RELEX	type-D,{dcb-addrx / (1)},{blkref-addrx / (0)}

type
specifies the access method; D specifies the direct-access method.

dcb
specifies the address of the data control block opened for the data set being processed.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

blkref
specifies the address of the block reference field.

If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

EXCEPTIONAL RETURNS: The return code in register 15 will be 04 if the block specified by blkref was not in exclusive status.

PROGRAMMING NOTES: A block is released from exclusive status by the RELEX macro-instruction or by being written back to the data set with the WRITE macro-instruction (by X in the type operand).

FREEDBUF -- Free Dynamically Obtained Buffer (R)

The FREEDBUF macro-instruction releases a buffer originally obtained by the control program in response to an 'S' in the area operand of a READ macro-instruction.

Name	Operation	Operand
[symbol]	FREEDBUF	{decb-addrx / (0)}, type-D, {dcb-addrx / (1)}

decb
specifies the address of a data event control block (DECB) used in a READ macro-instruction that specified an 'S' in the area operand.

If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

type
specifies the access method; D specifies the direct-access method.

dcb
specifies the address of the data control block opened for the data set being processed.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

CAUTIONS: The area field in the data event control block must not be altered between execution of the READ macro-instruction and the FREEDBUF macro-instruction.

The buffer can also be released by a WRITE macro-instruction.

Message Processing Routines

The queued telecommunication access method (QTAM) allows the user to obtain messages from, and place messages on, a message queue. These functions aid the message processing routines in data communications systems. The user must provide all the functions required in his message processing routine. These functions are described in detail in the publication IBM System/360 Operating System: Telecommunications, Form C28-6553. The message queue processing routines of the control program can be requested by means of the macro-instructions described in this section.

The main communication between a user's message processing routines and the message control routines of QTAM is the process program message queue. Incoming messages are stored in input (process) queues, and outgoing messages are stored in output (destination) queues. Both message control and message processing routines have access to these queues.

A user of QTAM must specify the manner in which messages are to be obtained from and placed into the queue. In both the input data control block and the output data control block, a user must specify the message processing routine's unit of data. The three options available are: record, segment, or complete message.

The GET and PUT macro-instructions are provided to gain access to message queues. In addition, the OPEN and CLOSE macro-instructions described in "General Service Macro-Instructions" must be used, as well as the DCB macro-instruction. The DCB, GET, and PUT macro-instructions are described in this section.

DCB -- Define QTAM Data Control Block

The DCB macro-instruction reserves space for a data control block required for a processing program message queue.

Name	Operation	Operand
[symbol]	DCB	DSORG=MQ,MACRF=(G|P) [,DDNAME=symbol][,TRMAD=relexp] [,SOWA=absexp][,RECFM=code] [,BUFRQ=absexp][,EODAD=relexp] [,EXLST=relexp][,SYNAD=relexp]

The keyword operands DSORG and MACRF can be supplied by only the DCB macro-instruction. The remaining operands can be supplied after assembly time by alternate sources; these alternate sources are indicated in the operand descriptions.

DSORG
specifies the data set organization; MQ is specified to indicate a processing program message queue for telecommunications.

MACRF
specifies the type of macro-instructions that will be used in referring to the data control block, as follows:

(G) - GET macro-instruction
(P) - PUT macro-instruction

DDNAME
specifies the name that appears in the DD statement that will be used to describe the data control block.

This information can also be supplied by the user's problem program before opening the data control block.

TRMAD
specifies the address of a user-provided area in which the terminal name is stored. When a GET macro-instruction is issued, QTAM places the source terminal name at the specified address. When a PUT macro-instruction is issued, the user must provide the destination terminal name at the specified address.

The only alternate source for this information is the user's problem program. If this operand is not supplied, the control program will terminate the task abnormally.

SOWA
specifies the size in bytes of the user-provided work area. The maximium value is 32,767.

This information can also be supplied by the DD statement or the user's problem program. If the work area size is not provided, the control program will terminate the task abnormally.

RECFM
specifies the work unit, as follows:

G - message (defined by the end-of-transmission character)
S - segment (defined by the buffer size)
R - record (defined by the carriage return, line field, combined carriage return and line feed, or end-of-block character)

This information can also be supplied by the DD statement or the user's problem program. If no value is provided, S is assumed.

BUFRQ
specifies the number of buffers to be read in advance from the direct-access device queue (before they are actually requested by a GET macro-instruction). This operand applies only to data control blocks for input queues on direct-access devices. The maximum value is 255.

This information can also be supplied by the DD statement or the user's problem program. If the value is not supplied, BUFRQ=0 is assumed.

EODAD
specifies the address of the user's end-of-data set exit routine for input data sets. This routine is entered when the user requests a record and there are no further records in the data set to be retrieved. If no routine has been provided, the task is abnormally terminated.

The only alternate source for this information is the user's problem program.

EXLST
specifies the address of the exit list. The format of the list is shown in appendix D.

Exit lists are required if the data control block exit is used.

The alternate source for this information is the user's problem program.

SYNAD
specifies the address of a user-provided routine to be entered if a work unit is longer than the work area provided.

The only alternate source for this information is the user's problem program. If it is not supplied, the remainder of the work unit will be supplied when the next GET macro-instruction is issued.

CAUTIONS: Separate data control blocks must be used for input and output.

The DCB macro-instruction cannot be written within the first 20 bytes of a control section. It can be preceded by padding, constants, or instructions.

GET -- Obtain Next Record (R)

The GET macro-instruction obtains the next sequential segment, record, or message from the input queue associated with the specified data control block. A reference to an empty queue will result in a wait condition unless a user's EODAD exit is provided in the data control block.

In the case of a polled terminal, the GET macro-instruction places the terminal name of the message source into the field specified by the TRMAD operand of the processing routine's input data control block. If the source terminal is not a polled type, the terminal name is placed into the field specified by TRMAD only if QTAM message control routines provide it. (Refer to the publication IBM System/360 Operating System: Telecommunications.)

Name	Operation	Operand
[symbol]	GET	{dcb-addrx / (1)}, {area-addrx / (0)}

dcb
specifies the address of the process program's input data control block that contains the parameters necessary to obtain work units from the desired queue.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

area
specifies the address of the area into which the desired segment, record, or message is placed.

If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

PROGRAMMING NOTE: The first four bytes of the work area are the record, segment, or message prefix. The first two of these bytes specifies the number of characters in the record, segment, or message. The third byte specifies the message type. (Refer to Table 32 for message type byte definitions.) The last byte of the prefix is a zero.

Table 32. Message Type Byte Definition Chart

Message Type Byte Contents	Message	Segment	Record
00000000		Header segment of a multiple-segment message	Header record of multiple-record message
00000001		Intermediate text segment	Intermediate text record
00000010	Complete message	Single segment complete message	Single record complete message
00000011		Last text segment of a multiple-segment message	Last text record of a multiple-record message

PUT -- Put Next Record (R)

The PUT macro-instruction places a message, segment, or record into an output queue. The terminal name of the message destination must be provided in the field whose address is specified by the DCBTRMAD field of the data control block.

Name	Operation	Operand
[symbol]	PUT	{dcb-addrx / (1)}, {area-addrx / (0)}

dcb
specifies the address of the output data control block that contains the parameters necessary to refer to the desired queue.

If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro-instruction.

area
specifies the address of the area from which the segment, record, or message is to be obtained by QTAM.

If (0) is written, the address must have been loaded into parameter register 0 before execution of this macro-instruction.

PROGRAMMING NOTES: The first four bytes in the work area must be the record segment or message prefix. The first two of these bytes must specify the number of characters in the record, segment, or message. The third byte must specify the message type (Table 32). The last byte of the prefix is a zero.

The Operating System/360 control program includes an advanced facility for execution-time testing of problem programs. Known as TESTRAN, for test translator, this facility performs test services specified by the programmer through macro-instructions included in his source program.

Services are performed at specified points in the problem program; the order of performance depends on the sequence of the TESTRAN macro-instructions. Macro-instructions and problem program instructions can be intermixed, grouped separately, or even assembled independently.

Services are performed by control program routines known collectively as the TESTRAN interpreter. The user has the option of conserving main storage by specifying that individual routines be called from external residence only when specifically required. Alternatively, the user can conserve test execution time by specifying that routines remain resident in main storage for the duration of the test.

Service routines are reenterable and may therefore be used by several tasks in multiprogrammed tests. These routines are protected against modification by the problem program, and include facilities to detect runaway testing and excessive test output.

The data produced by the TESTRAN interpreter is retained as a data set for processing by the TESTRAN editor. The data can be recorded either on tape or on a direct-access device, and can be selected for printing either in its entirety or according to output selection codes expressed in the related macro-instructions. The test data, including all associated symbols, is printed in the format defined in the source program.

Test Services: Services provided by the TESTRAN macro-instructions include both action and control functions. Test actions include the dumping (recording for display) of system tables, registers, and main storage and the tracing of transfers, subroutine calls, and references to data. Control capabilities include the dynamic testing of conditions resulting from program execution; performance of test actions can accordingly be made dependent on conditions detected during processing. When an error condition is detected in this manner, an attempt at recovery can be made by alteration of problem program data or control flow. Specific test actions can be organized as subroutines or executed repetitively under loop control. Test action and control capabilities are described in "TESTRAN Macro-Instructions."

Test Procedure: Required job steps - assembly, linkage editing, testing, and editing of test data - can be performed either as separate jobs or as steps within a single job. The programmer has the ability to assemble parts of his program independently and to correct assembly errors before proceeding with the test. He is also able to repeat the editing of output from a single test, selecting only data of immediate interest on each repetition. Use of system facilities and options is described further in "Job Organization."

TESTRAN OPERATION

TESTRAN macro-instructions and the problem program to be tested are first assembled, either together or separately. The assembler, upon request, produces a symbol table in addition to its normal output; this table contains the symbolic names, the data attributes (type, length, and scale), and the attribute of named instructions (type) for the problem program. The symbol table is processed by the linkage editor together with the assembled problem program, the macro-instructions, and the control dictionaries. It is used during post-processing to edit test data into a format that includes the symbolic names and data attributes of the source program.

CSECT assembler instructions are generated at the beginning and end of the macro-expansion of each TESTRAN macro-instruction. As a result, the macro-expansions are always loaded out-of-line as one or more separate control sections, even if the macro-instructions are scattered among the instructions of the problem program. A separate control section is generated for each TEST OPEN macro-instruction; it contains the corresponding macro-expansion plus the macro-expansions of the TESTRAN macro-instructions that follow the TEST OPEN (and precede any subsequent TEST OPEN) in the source program. Each control section contains only a single executable instruction, an SVC; the address of this instruction and the name of the control section are both specified by the symbolic name of the corresponding TEST OPEN macro-instruction.

When control is passed to a TEST OPEN macro-instruction, the TESTRAN interpreter initiates testing by inserting SVC instructions at specified addresses in the problem program. These addresses are specified in TEST AT macro-instructions and represented as constants in the corresponding macro-expansions. Each instruction overwritten by insertion of an SVC is saved, and control is returned to the problem program.

When the control flow of the problem program causes execution of an inserted SVC instruction, control is passed to the TESTRAN interpreter routing routine. This routine causes TESTRAN macro-instructions to be encountered in a logical sequence similar to that of an executable program. That is, it provides linkage to service routines that interpret the nonexecutable macro-expansions and perform services specified in the corresponding macro-instructions of the source program. TESTRAN macro-instructions are encountered sequentially, beginning with the first macro-instruction following the TEST AT macro-instruction that caused control to be given to the routing routine. However, the sequence can be altered, either conditionally or unconditionally, by certain macro-instructions. When a GO BACK macro-instruction is encountered, the instruction overwritten by insertion of the SVC instruction is executed, and control is returned to the problem program.

The TESTRAN editor processes recorded test data and causes that produced by specified macro-instructions to be printed on the system output device. The editor program is executed as an independent job step and is controlled entirely by job control statements.

Note: A program under test is not reusable. A single copy of the program can be reentered only by direct (type I) linkage, because a supervisor-assisted (type II) linkage always causes loading of a new copy. Direct linkage must not be made from programs operating under more than one task.

TESTRAN MACRO-INSTRUCTION STATEMENT FORMAT

The statement format for TESTRAN macro-instructions is the same as for other system macro-instructions: it includes a name field, a mnemonic operation code, and one or more operands. The name field can be blank or can contain a symbol. A symbolic name must always be assigned to the TEST OPEN macro-instruction to provide a name for the control section containing the corresponding macro-expansion. Symbolic names can optionally be assigned to other TESTRAN macro-instructions; these names can be referred to by TESTRAN macro-instructions but should not be referred to by the problem program.

Each TESTRAN macro-instruction includes a mnemonic operation code that specifies the general type of test service to be performed. The five TESTRAN macro-instructions are designated by the operation codes DUMP, TRACE, TEST, GO, and SET. DUMP and TRACE perform actions that produce output related to the action of the problem program; they are accordingly referred to as "action" macro-instructions. TEST, GO, and SET perform functions required for control of test operation; they are therefore known as "control" macro-instructions.

In addition to the operation code, each TESTRAN macro-instruction must include an initial coded value operand that specifies the particular type of test service to be performed. Designated by operation code and first operand, the twenty-three specific forms of the TESTRAN macro-instructions, together with their basic usage, are as listed in Table 33. Each of these forms is described fully in "TESTRAN Macro-Instructions."

Additional operands are described in the detailed descriptions of the specific forms of the TESTRAN macro-instructions. Keyword operands are always optional. The most common types and their general usage are shown in Table 34.

VALUE MNEMONICS: The following value mnemonics are used in the descriptions of TESTRAN macro-instructions.

- symbol
- relexp
- addx
- adval
- integer
- text
- tls

These value mnemonics are defined in Section 1 of this publication. The operand forms indicated by them are discussed in detail in Appendix A, and should be reviewed before operands are written.

TESTRAN MACRO-INSTRUCTIONS

This section contains a detailed description of each TESTRAN macro-instruction and a set of notes on the usage of certain macro-instructions and their operands.

Table 33. Forms of the TESTRAN Macro-Instructions

Macro-Instruction (specific form)	Usage
DUMP DATA	Records the contents of a specified area of main storage.
DUMP CHANGES	Records the contents of a specified area of main storage and identifies successive changes.
DUMP MAP	Records addresses of control sections and dynamically allocated storage areas.
DUMP TABLE	Records the contents of a specified system table.
DUMP PANEL	Records the contents of the program status word and of specified general and floating-point registers.
DUMP COMMENT	Records a comment written by the programmer.
TRACE FLOW	Indicates each execution of a program transfer to, from, or within a specified area of main storage.
TRACE CALL	Indicates each execution of a CALL macro-instruction located within a specified area of main storage.
TRACE REFER	Indicates the execution and effect of each reference to data within a specified area of main storage.
TRACE STOP	Stops traces initiated by specified TRACE FLOW, TRACE CALL, and TRACE REFER macro-instructions.
TEST OPEN	Initiates program testing and provides overall test control.
TEST AT	Specifies points within the problem program at which testing is to occur.
TEST DEFINE	Defines flags and counters used by TEST WHEN, TEST ON, SET FLAG, and SET COUNTER macro-instructions.
TEST WHEN	Controls the sequence in which other test services are performed by testing for a specified logical condition or arithmetic relationship.
TEST ON	Controls the sequence in which other test services are performed by incrementing a counter and testing it for specified values.
TEST CLOSE	Terminates program testing.
GO TO	Causes test operation to continue with a specified TESTRAN macro-instruction.
GO IN	Saves the address of the next sequential macro-instruction and causes test operation to continue with a specified TESTRAN macro-instruction.
GO OUT	Causes test operation to continue with the macro-instruction that follows an associated GO IN macro-instruction.
GO BACK	Ends a sequence of test services and returns control to the problem program.
SET FLAG	Assigns a condition to a specified logical flag.
SET COUNTER	Assigns a value to a specified program-testing counter.
SET VARIABLE	Assigns a value to a specified register or data item in the problem program.

Table 34. Common Keyword Operands and Their Usage

Keyword	Usage
SELECT	Specifies an output selection code by which associated test data can be selected for editing.
DATAM	Specifies data attributes to be used in place of those defined in the symbol table.
NAME	Specifies a symbol to be printed with test data generated by DUMP macro-instructions. This symbol is printed in place of symbolic names defined in the symbol table.
COMMENT	Specifies a comment to be printed with test data generated by a TRACE macro-instruction.
DSECT	Identifies address operands as referring to a dummy control section.

MACRO-INSTRUCTION DESCRIPTIONS

All macro-instructions described in this section are identified by a three-digit macro-instruction identification number (macro ID) that is automatically assigned by the assembler. This number is printed with each macro-instruction in the assembly listing; it is included in the printed test data to identify the sequence in which test services have been performed and the resulting output.

A TESTRAN macro-instruction must not be placed among the instructions or data of a dummy control section. If this restriction is violated, the last statement in the macro-expansion will be a CSECT assembler instruction that has the same name as the preceding DSECT assembler instruction. The CSECT instruction will be treated as invalid, and subsequent instructions or data will be placed in an unnamed control section.

DUMP DATA -- Record Main Storage

This form of the DUMP macro-instruction records the contents of a specified area of main storage. All resulting test data, when selected for editing, is printed with the attributes and symbolic names defined in the source program symbol table.

Name	Operation	Operand
[symbol]	DUMP	DATA,start-addx[,end-addx] [,SELECT=integer] [,DATAM=tls] [,NAME=symbol] [,DSECT=(dsect-symbol[,repeat-integer])]

DATA
specifies the DUMP DATA form of the DUMP macro-instruction.

start
specifies the starting address of the storage area; if the end operand is omitted, the single data item or instruction starting at the starting address is recorded.

end
specifies the ending address of the storage area; if this operand is present, the main storage area from the starting address to, but not including, the ending address is recorded.

SELECT
specifies an output selection code (an integer from 1 to 8) by which the recorded test data can be selected for editing.

DATAM
specifies the data attributes (type, length, scale) to be used when the recorded test data is selected for editing. These attributes are used in place of attributes defined in the symbol table.

NAME
specifies a symbolic name to be printed with the recorded test data; names defined in the symbol table for the same data are not printed.

DSECT
indicates that the starting address and, if present, the ending address are associated with the dummy control section specified by the dsect operand. If present, the repeat operand specifies a decimal integer by which the length of the specified storage area is multiplied at execution time; if absent, repeat has an assumed value of 1. The effective length of the storage area is the product of this multiplication. When the repeat factor causes an extension of the specified storage area, the section of the symbol table associated with the original range of addresses is used repetitively to provide symbolic names and data attributes for the extended area.

EXAMPLES: In the following examples, EX1 records the contents of the main storage area beginning at FIELD7. The length of this field is as defined by the symbol table.

EX2 records the contents of main storage from AREA1 through AREA2-1. The data is printed when test data with an output selection code of 1 is selected for editing; printed output is in fixed-point format.

EX3 records the contents of the first 500 bytes of a dummy control section named BUFFER. Each block of 50 bytes is edited according to the data attributes contained in the symbol table for the addresses BUFFER through BUFFER+49. Printing of symbolic names for addresses in this range is suppressed; the name INPUT is printed once as a label applied to the contents of the full 500 bytes.

```
EX1      DUMP  DATA,FIELD7
EX2      DUMP  DATA,AREA1,AREA2,SELECT=1,DATAM=F
EX3      DUMP  DATA,BUFFER,BUFFER+50,NAME=INPUT,DSECT=(BUFFER,10)
```

PROGRAMMING NOTES: When both starting and ending addresses are specified, the range of a main storage dump is determined by the corresponding loaded addresses. In a scatter-loaded problem program, this range may vary unpredictably if starting and ending addresses are in separate control sections. The following conditions may occur:

- Control sections that were not part of the range in the source program may be included in the range of loaded addresses.
- Control sections that were a part of the original range may be omitted from the range of loaded addresses.
- The control section containing the starting address may be loaded at a storage location higher numbered than the control section containing the ending address. If this situation occurs, only the single byte at the starting address is recorded, and a diagnostic message is inserted in the printed test output each time the macro-instruction is encountered.

A program that is scatter loaded should therefore include a separate macro-instruction for each control section to be recorded.

When the range includes more than one control section, only data from the first control section can be edited into the format defined by the symbol table. Unless a DATAM or DSECT operand was written, data from other control sections is printed in 4-byte hexadecimal format.

The maximum range for a storage dump is 65,535 bytes. A dump is truncated if the specified ending address exceeds the starting address by more than 65,535.

The range of a dump must not include storage areas that have been assigned to different tasks. If it does, the dump is truncated to include only contiguous areas associated with the current task.

Use of the linkage editor CHANGE control statement to change the names of external symbols does not affect symbolic names that appear in test output. Use of this facility to change the names of control sections causes dumps of these control sections to be printed in hexadecimal format unless overriding data attributes are specified.

DUMP CHANGES -- Record Main Storage and Identify Changes

This form of the DUMP macro-instruction records the contents of a specified area of main storage. All resulting test data, when selected for editing, is printed with the attributes and symbolic names defined in the source program symbol table. The initial recording is printed in its entirety. When subsequent records produced by the same DUMP CHANGES macro-instruction are edited, only those fields that have changed since the previous recording are printed. If the range between the starting and ending addresses is altered by indexing, the full contents of any addition to the previous range are printed, and only that portion of the range included in the previous range is examined for changes.

Name	Operation	Operand
[symbol]	DUMP	CHANGES,start-addx[,end-addx] [,SELECT=integer] [,DATAM=tls] [,NAME=symbol] [,DSECT=(dsect-symbol[,repeat-integer])]

CHANGES
specifies the DUMP CHANGES form of the DUMP macro-instruction.

start
specifies the starting address of the storage area; if the end operand is omitted, the single data item or instruction starting at the starting address is recorded and examined for changes.

end
specifies the ending address of the storage area; if this operand is present, the main storage area from the starting address to, but not including, the ending address is recorded and examined for changes.

SELECT
specifies an output selection code (an integer from 1 to 8) by which the recorded test data can be selected for editing.

DATAM
specifies the data attributes (type, length, scale) to be used when the recorded test data is selected for editing. These attributes are used in place of the attributes defined in the symbol table.

NAME
specifies a symbolic name to be printed with the recorded test data; names defined in the symbol table for the same data are not printed.

DSECT
indicates that the starting address and, if present, the ending address are associated with the dummy control section specified by the dsect operand. If present, the repeat operand specifies a decimal integer by which the length of the specified storage area is multiplied at execution time; if absent, repeat has an assumed value of 1. The effective length of the storage area is the product of this multiplication. When the repeat factor causes an extension of the specified storage area, the section of the symbol table associated with the original range of addresses is used repetitively to provide symbolic names and data attributes for the extended area.

EXAMPLE: In the following example, EX1 records the contents of the storage area from CONSTANT through CONSTANT+79. The initial recording of this area is printed as 10 hexadecimal fields of 8 bytes each. Subsequent recordings are also edited as 8-byte hexadecimal fields, but only those fields that contain changes will be printed. The name CONSTANT is printed with the related data, as are other symbolic names included in the symbol table for this area.

```
EX1      DUMP  CHANGES,CONSTANT,CONSTANT+80,DATAM=XL8
```

PROGRAMMING NOTES: To edit test data produced by DUMP CHANGES macro-instructions, the TESTRAN editor must scan and compare storage records contained in an intermediate data set. This process is time consuming and, if many such records exist, considerably reduces the speed of editing. The number of DUMP CHANGES macro-instructions used and the size of storage areas recorded should accordingly be carefully limited.

The programming notes for DUMP DATA apply also to DUMP CHANGES macro-instructions.

DUMP MAP -- Record Storage Map

This form of the DUMP macro-instruction records the names, addresses, and lengths of all control sections and dynamically allocated storage areas associated with the task being executed.

Name	Operation	Operand
[symbol]	DUMP	MAP[,SELECT=integer]

MAP
specifies the DUMP MAP form of the DUMP macro-instruction.

SELECT
specifies an output selection code (an integer from 1 to 8) by which the recorded test data can be selected for editing.

EXAMPLE: In the following example, EX1 records a map of control sections and dynamically allocated storage associated with the current task. This map is printed when test data with no associated output selection code is selected for editing.

```
EX1      DUMP  MAP
```

DUMP TABLE -- Record System Table

This form of the DUMP macro-instruction records the contents of a specified system table. When selected for editing, the contents of the table are printed in a meaningful format determined by the system.

Name	Operation	Operand
[symbol]	DUMP	TABLE,{block-{DCB|DEB},dcb-addx / block-TCB} [,SELECT=integer]

TABLE
specifies the DUMP TABLE form of the DUMP macro-instruction.

block
specifies the type of system table to be recorded, as follows:

DCB - data control block
DEB - data extent block
TCB - task control block

If TCB is present, the recorded task control block is that associated with the current task.

dcb
specifies the address of a data control block. If DCB is present, the dcb operand specifies the data control block that is to be recorded. If DEB is present, the dcb operand specifies the data

control block associated with the data extent block that is to be recorded.

SELECT
specifies an output selection code (an integer from 1 to 8) by which the recorded test data can be selected for editing.

EXAMPLE: In the following example, EX1 causes the contents of the data control block named TESTDCB to be printed in a predetermined format when test data with an output selection code of 8 is selected for editing.

```
EX1      DUMP  TABLE,DCB,TESTDCB,SELECT=8
```

DUMP PANEL -- Record Registers and PSW

This form of the DUMP macro-instruction records the contents of the program status word and of specified general and floating-point registers. Registers are printed in 4-byte hexadecimal format.

Name	Operation	Operand
[symbol]	DUMP	PANEL[,({register-ttrn,}...)]
		[,SELECT=integer]
		[,DATAM=tls]

PANEL
specifies the DUMP PANEL form of the DUMP macro-instruction.

register
specifies a register or series of registers to be recorded; if this operand is omitted, all registers are recorded. The value mnemonic ttrn indicates that this operand is to be written in test translator register notation. For a full description of this notation, including its use to specify a series of registers, refer to Appendix A.

SELECT
specifies an output selection code (an integer from 1 to 8) by which the recorded test data can be selected for editing.

DATAM
specifies the data attributes (type, length, scale) to be used when the recorded test data is selected for editing. These attributes are used in place of the implicit attributes (hexadecimal type and 4-byte length) and apply to all registers in the sublist.

EXAMPLES: In the following examples, EX1 records the contents of the program status word and general register 7. The program status word is printed in a standard format; the contents of the register are printed as an 8-digit hexadecimal field.

EX2 records the contents of the program status word and the floating-point registers 0, 2, and PRODUCT. (PRODUCT is a symbol equated to the number of a floating-point register.) The program status word is printed in a standard format; the contents of the floating-point registers are printed in long floating-point notation.

```
EX1      DUMP   PANEL,G'7'
EX2      DUMP   PANEL,(F'0,2',F'PRODUCT'),DATAM=D
```

DUMP COMMENT -- Record Comment

This form of the DUMP macro-instruction records a specified comment for printing.

Name	Operation	Operand
[symbol]	DUMP	COMMENT,commentfield-'text'[,SELECT=integer]

COMMENT
specifies the DUMP COMMENT form of the DUMP macro-instruction.

commentfield
specifies the comment to be recorded.

SELECT
specifies an output selection code (an integer from 1 to 8) by which the recorded comment can be selected for editing.

EXAMPLE: In the following example, EX1 records the comment specified. It is printed when test data with an output selection code of 4 is selected for output.

```
EX1      DUMP  COMMENT,'EXPERIMENTAL ERROR EXCEEDS ALLOWABLE LIMIT --
               THEORETICAL VALUE SUBSTITUTED FOR K AND PROCESSING
               RESUMED',SELECT=4
```

TRACE FLOW -- Record Program Transfers

This form of the TRACE macro-instruction indicates each execution of a program transfer (branch or SVC) to, from, or within a specified area of main storage. The instruction that executes the transfer is recorded, as are the addresses from and to which the transfer is made, and the condition code at the time of the transfer. When a branch instruction is executed, the contents of all registers used by the instruction are recorded. When an SVC instruction is executed, the contents of general registers 0 and 1 are recorded. The execution of an EX instruction is recorded if it causes execution of a program transfer.

Name	Operation	Operand
[symbol]	TRACE	FLOW,start-addx[,end-addx] [,SELECT=integer] [,COMMENT='text']

FLOW
specifies the TRACE FLOW form of the TRACE macro-instruction.

start
specifies the starting address of the storage area; if the end operand is omitted, only program transfers to or from the starting address are recorded.

end
specifies the ending address of the storage area; this area includes all locations from the starting address to, but not including, the ending address. All program transfers to, from, or within this area are recorded.

SELECT
specifies an output selection code (an integer from 1 to 8) by which the recorded test data can be selected for editing.

COMMENT
specifies a programmer-written comment to be printed with the recorded test data.

EXAMPLE: In the following example, EX1 records all program transfers to, from, or within the area from DECISION through NOBRANCH-1. The specified comment is printed with each indication of an executed transfer.

```
EX1      TRACE FLOW,DECISION,NOBRANCH,COMMENT='PROBLEM PROGRAM BRANCHES
               FROM DECISION AREA'
```

PROGRAMMING NOTES: Tracing activity requires the examination of each problem program instruction executed while the trace is active. This process is time consuming, but the time required can be minimized by limiting the duration of the trace and the output the trace produces. The programmer can limit the duration of the trace by means of TRACE STOP macro-instructions; when encountered at execution time, these macro-instructions suspend tracing activities specified as being of no current interest to the programmer. The programmer can limit the output of the trace, within a general area of main storage, to only those areas of special interest by specifying each such area in a separate macro-instruction.

No more than ten traces (corresponding to ten TRACE macro-instructions) can be active simultaneously. An attempt to start an eleventh trace will cause the tenth trace -- the trace most recently started -- to be suspended.

The programmer must consider the following facts when interpreting the output of trace routines:

- Tracing is performed only in storage areas associated with the problem program or with problem state control program routines. Tracing is suspended when supervisor-state control program routines receive control, and is resumed when the control program relinquishes control.
- A trace is suspended on overlay of the segment containing the corresponding TRACE macro-instruction. It is not automatically resumed when the segment is reentered.
- A subroutine or program segment must contain its own TRACE macro-instructions if it receives control from the control program through an asynchronous interruption (other than a program check) or through an ATTACH, LINK, or XCTL macro-instruction. All traces active for a task are suspended upon linkage by an ATTACH, LINK, or XCTL macro-instruction, and upon execution of a RETURN macro-instruction terminating a task or program that receives control through one of these macro-instructions.

The range of addresses in which tracing is recorded is determined by the loaded addresses corresponding to the address operands. In a problem program that is scatter loaded, this range may vary unpredictably if the starting and ending addresses are in separate control sections. The following conditions may occur:

- Control sections that were not part of the range in the source program may be included in the range of loaded addresses.
- Control sections that were a part of the original range may be omitted from the range of loaded addresses.
- The control section containing the starting address may be loaded at a storage location higher numbered than the control section containing the ending address. If this situation occurs, the macro-instruction is ignored and a diagnostic message inserted in the test output each time the macro-instruction is encountered.

A program that is scatter loaded should therefore include a separate macro-instruction for each control section in which traces are to be recorded.

The range of addresses for a given macro-instruction can be varied by indexing. A new trace is started each time the macro-instruction is encountered, and any trace already active for that macro-instruction is automatically suspended.

TRACE CALL -- Record Execution of CALL Macro-Instructions

This form of the TRACE macro-instruction indicates each execution of a CALL macro-instruction that is located in a specified area of main storage. The symbolic name and the assembled and loaded addresses of each CALL and called routine are recorded together with the contents of all registers used by the CALL.

Name	Operation	Operand
[symbol]	TRACE	CALL,start-addx,end-addx [,SELECT=integer] [,COMMENT='text']

CALL
specifies the TRACE CALL form of the TRACE macro-instruction.

start
specifies the starting address of the storage area within which the execution of CALL macro-instructions is to be recorded.

end
specifies the ending address of the storage area; this area includes all locations from the starting address to, but not including, the ending address.

SELECT
specifies an output selection code (an integer from 1 to 8) by which the recorded test data can be selected for editing.

COMMENT
specifies a programmer-written comment to be printed with the recorded test data.

EXAMPLE: In the following example, EX1 records the execution of all CALL macro-instructions located in the area from SUBRTNE1 through SUBRTNE5-1. The symbolic name and the assembled and loaded addresses of each CALL and called routine (as well as the contents of registers used by the CALL) are printed when test data with an output selection code of 3 is selected for editing.

```
EX1       TRACE CALL,SUBRTNE1,SUBRTNE5,SELECT=3
```

PROGRAMMING NOTES: Refer to the programming notes for TRACE FLOW.

TRACE REFER -- Record Storage References

This form of the TRACE macro-instruction indicates each reference by problem program instructions that could change data within a specified area of main storage. The instruction making the reference is recorded, as are the values of the data before and after the reference, the addresses of the instruction and the data, and the contents of any related registers. The execution of an EX instruction is recorded if it causes a reference to be made.

Name	Operation	Operand
[symbol]	TRACE	REFER,start-addx[,end-addx] [,SELECT=integer] [,DATAM=t|s] [,COMMENT='text'] [,DSECT=(dsect-symbol[,repeat-integer])]

REFER
specifies the TRACE REFER form of the TRACE macro-instruction.

start
specifies the starting address of the storage area; if the end operand is omitted, only references to the field beginning at the starting address are recorded.

end
specifies the ending address of the storage area; this area includes all locations from the starting address to, but not including, the ending address. All references to this area are recorded.

SELECT
specifies an output selection code (an integer from 1 to 8) by which the recorded test data can be selected for editing.

DATAM
specifies the data attributes (type, length, scale) to be used when the recorded test data is selected for editing. These attributes are used in place of the attributes defined in the symbol table.

COMMENT
specifies a programmer-written comment to be printed with the recorded test data.

DSECT
indicates that the starting address and, if present, the ending address are associated with the dummy control section specified by the dsect operand. If present, the repeat operand specifies a decimal integer by which the length of the specified storage area is multiplied at execution time; if absent, repeat has an assumed value of 1. The effective length of the storage area is the product of this multiplication. When the repeat factor causes an extension of the specified storage area, the section of the symbol table for the original range of addresses is used repetitively to provide symbolic names and data attributes for the extended area.

EXAMPLE: In the following example, EX1 records all modifications of the data item DIVIDEND. The values of DIVIDEND before and after the reference are each printed as fields containing 7 decimal digits and a sign. The specified comment is printed with each reference indicated by EX1.

```
EX1      TRACE REFER,DIVIDEND,COMMENT='QUOTIENT AND REMAINDER RESULTING
               FROM DECIMAL DIVISION OF DIVIDEND',DATAM=PL4
```

PROGRAMMING NOTES: The range of addresses specified in a TRACE REFER macro-instruction must not include addresses 0 to 48. Results are unpredictable if this restriction is violated.

When the range includes more than one control section, only data from the first control section can be edited into the format defined in the symbol table. Unless a DATAM or DSECT operand was written, data from other control sections is printed in 4-byte hexadecimal format.

The programming notes for TRACE FLOW apply also to TRACE REFER macro-instructions.

TRACE STOP -- Suspend Traces

This form of the TRACE macro-instruction suspends tracing activities previously initiated by specified TRACE FLOW, TRACE CALL, and TRACE REFER macro-instructions. The macro-instruction identification number of each specified macro-instruction and the symbolic name of the control section in which each is located are both recorded. Suspended activities are restarted if the initiating macro-instructions are subsequently reencountered.

Name	Operation	Operand
[symbol]	TRACE	STOP[,({macro-symbol,}...)][,SELECT=integer]

STOP
specifies the TRACE STOP form of the TRACE macro-instruction.

macro
specifies the symbolic name of a TRACE macro-instruction that initiated a tracing activity to be suspended; any number of symbolic names can be specified. If the sublist is omitted, all current tracing activity is suspended.

SELECT
specifies an output selection code (an integer from 1 to 8) by which the related test data can be selected for editing.

EXAMPLE: In this example, EX1 suspends tracing activities initiated by the TRACE macro-instructions named TR2 and TR5.

```
EX1       TRACE STOP,(TR2,TR5)
```

TEST OPEN -- Initiate Testing

This form of the TEST macro-instruction initiates the performance of test services when it receives control either from the control program (e.g., as a program entry point) or from the problem program. Although it is the only executable TESTRAN macro-instruction, it is ignored if encountered following a reference by a TEST WHEN, TEST ON, GO TO, or GO IN macro-instruction. Note that a TEST OPEN macro-instruction must always be given a symbolic name, and must be the first TESTRAN macro-instruction encountered during assembly. For a detailed description of the use of this macro-instruction, refer to "Notes on Usage."

Name	Operation	Operand
symbol	TEST	OPEN[,entry-relexp,[ident-symbol] [,linkage-{LINK|LOAD}]] [,MAXP=integer] [,MAXE=integer] [,OPTEST=({macro-symbol,}...)] [,SELECT=integer]

OPEN
specifies the TEST OPEN form of the TEST macro-instruction.

entry
specifies an address in the problem program; control is returned to this address after execution of the TEST OPEN routine, and the address is placed in register 15. This operand is required when a TEST OPEN macro-instruction either is the problem program entry point or receives control through a branch executed by the problem program. Omission under these conditions results in an abnormal end of task. This operand is not required if the TEST OPEN macro-instruction is not executed but, instead, is referred to by the OPTEST operand of another TEST OPEN macro-instruction.

ident
specifies a symbol that is to be included in a standard page heading to identify printed test output.

linkage
specifies the system macro-instruction that is to provide linkage to test translator service routines as they are required during execution. LINK minimizes storage requirements but may cause slower operation; LOAD maximizes operating speed but requires a larger area of main storage. If neither option is specified, the LINK option is assumed.

MAXP
specifies the maximum number of pages of test data to be produced. This limit is approximate and must not be greater than the limit established at the installation during system generation. If either the programmer's or the installation's limit is exceeded, an appropriate message is inserted in the test output and an abnormal end of task results.

MAXE
specifies the maximum number of TESTRAN macro-instructions to be encountered, counting each macro-instruction once for each time that it is encountered. This number must not exceed the limit established at the installation during system generation. If either the programmer's or the installation's limit is exceeded, an appropriate message is inserted in the test data and an abnormal end of task results.

OPTEST
specifies the symbolic names of other TEST OPEN macro-instructions. These macro-instructions initiate additional testing without receiving control directly from either the control program or the problem program. Any optional operands included in these TEST OPEN macro-instructions are ignored.

SELECT
specifies an output selection code (an integer from 1 to 8) by which test data with no other associated output selection code can be selected for editing. This code applies to action macro-instructions that follow this TEST OPEN (and precede any subsequent TEST OPEN) in the source program. This code overrides the output selection codes specified in the TEST OPEN macro-instructions specified in the OPTEST operand; it is overridden by any other output selection code associated with action or TEST AT macro-instructions.

EXAMPLE: In the following example, testing specified by macro-instructions following TEST1 and TEST2 in the source program is initiated when control is passed to TEST1; control is subsequently returned to the address START in the problem program. Test output is limited either to 75 pages or to the output of 20 encountered TESTRAN macro-instructions, whichever is the lesser quantity. All printed pages are headed with the label TWOTESTS; all test data with no other associated output selection code has an implicit output selection code of 8.

```
TEST1    TEST  OPEN,START,TWOTESTS,MAXP=75,MAXE=20,OPTEST=TEST2,SELECT=8
           .     .
           .     .
           .     .
TEST2    TEST  OPEN
           .     .
           .     .
           .     .
```

TEST AT -- Perform Testing at Problem Program Address

This form of the TEST macro-instruction specifies points within the problem program at which testing is to occur. The test services performed are those specified by the series of TESTRAN macro-instructions that begins with the next sequential TESTRAN macro-instruction. If a TEST AT macro-instruction is encountered in this series, it functions as a GO BACK macro-instruction with no return operand.

Name	Operation	Operand
[symbol]	TEST	AT,({address-{relexp|*},}...)[,SELECT=integer]

AT
specifies the TEST AT form of the TEST macro-instruction.

address
specifies a problem program address at which a sequence of test services is to be performed; any number of addresses can be specified. If written as an asterisk, the operand specifies the current value of the location counter for the problem program control section that contains the last encountered assembler language instruction.

SELECT
specifies an output selection code (an integer from 1 to 8) by which test data with no other associated output selection code can be selected for editing. This code applies to action macro-instructions that follow the TEST AT (and precede any subsequent TEST AT or TEST OPEN) in the source program. This code overrides any code specified in a related TEST OPEN macro-instruction; it is overridden by output selection codes specified in the action macro-instructions themselves.

EXAMPLE: In the following example, when the TEST OPEN macro-instruction named INIT returns control to BEGIN in the problem program, the TEST AT macro-instruction causes test services specified by a series of macro-instructions (not shown) to be performed. An output selection code of 4 is assigned to test data produced by following macro-instructions in which no output selection code is specified.

```
INIT     TEST  OPEN,BEGIN,SELECT=8
         TEST  AT,BEGIN,SELECT=4
         .     .
         .     .
         .     .
```

PROGRAMMING NOTES: When testing is initiated by a TEST OPEN macro-instruction, the TESTRAN interpreter inserts an SVC instruction at each problem program address specified by a TEST AT macro-instruction. Test services are performed whenever the inserted SVC's are executed as part of the problem program. Each address specified must therefore be the address of an executable instruction, and each of these instructions must adhere to the following restrictions:

1. The instruction must not be a privileged instruction. If this restriction is violated, execution of the privileged instruction results in an abnormal end of task when the TESTRAN interpreter returns control to the problem program.

2. The instruction must not be modified by any instruction in the problem program. Results are unpredictable if this rule is violated.

3. The instruction must not be an SVC instruction; if it is an SVC instruction, the TESTRAN SVC is not inserted.

4. The instruction must not be an EX instruction that causes execution of an instruction that violates rule 1, 2, or 3.

5. The instruction must not be executed by an EX instruction. If this rule is violated, test services are not performed; an error message is inserted in the test output and the EX instruction is ignored.

TEST DEFINE -- Define Flags or Counters

This form of the TEST macro-instruction specifies that either logical flags or program-testing counters are to be assembled for use by TEST WHEN, TEST ON, SET FLAG, and SET COUNTER macro-instructions. Flags and counters are assembled as data items in the control section that contains the TEST DEFINE. This macro-instruction has no function at execution time and, if encountered, is ignored.

Name	Operation	Operand
[symbol]	TEST	DEFINE,{FLAG,({flag-symbol,}...) / COUNTER,({ctr-symbol,}...)}

DEFINE
specifies the TEST DEFINE form of the TEST macro-instruction.

FLAG
specifies that one or more logical flags are to be assembled, each with an initial condition of 0.

flag
specifies a symbolic name for a logical flag that is to be assembled. Any number of symbolic names can be specified; each causes assembly of a separate logical flag and must be unique within the object module.

COUNTER
specifies that one or more counters are to be assembled, each with an initial value of 0.

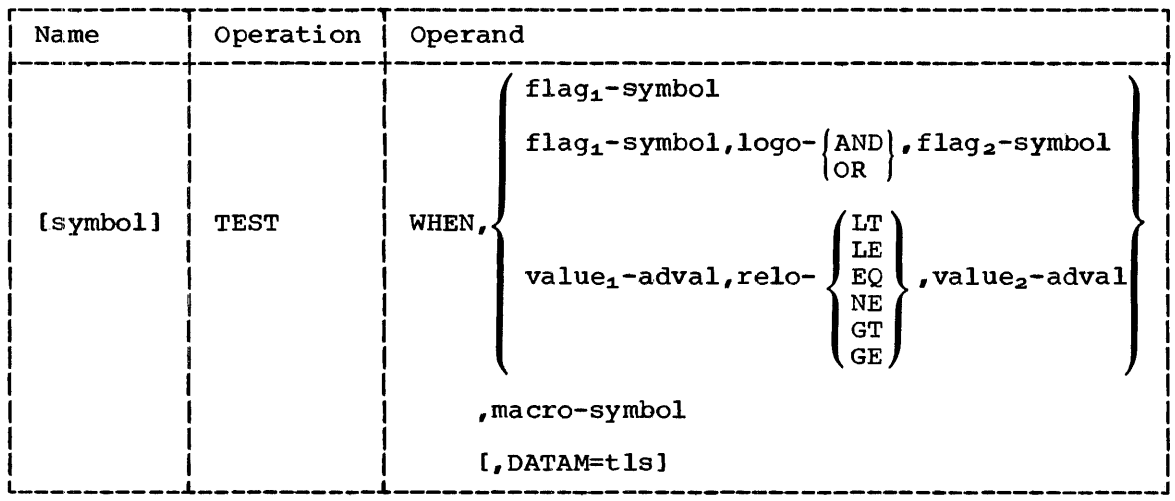
ctr
specifies a symbolic name for a counter that is to be assembled. Any number of symbolic names can be specified; each causes assembly of a separate counter and must be unique within the object module.

EXAMPLES: In the following examples, EX1 defines logical flags with initial conditions of 0 and symbolic names RED, BLUE, and GREEN. EX2 defines a counter with an initial value of 0 and the symbolic name LOOPCNT.

```
EX1     TEST  DEFINE,FLAG,(RED,BLUE,GREEN)
EX2     TEST  DEFINE,COUNTER,LOOPCNT
```

TEST WHEN -- Alter Test Sequence When Condition or Relationship Occurs

This form of the TEST macro-instruction controls the sequence in which other test services are performed by testing for a logical condition or arithmetic relationship each time the macro-instruction is encountered at execution time. An affirmative test result causes the TESTRAN interpreter to perform the services specified by a series of TESTRAN macro-instructions, the first of which is specified by its symbolic name. A nonaffirmative test esult causes the TESTRAN interpreter to perform the test services specified by the series of macro-instructions that begins with the next sequential macro-instruction.



WHEN
specifies the TEST WHEN form of the TEST macro-instruction.

$flag_1$
specifies the name of a logical flag defined by a TEST DEFINE macro-instruction. If no other logical flag is specified, a test is affirmative when the specified flag has a condition of 1.

logo
specifies the logical operator AND or OR.

$flag_2$
specifies the symbolic name of a second logical flag defined by a TEST DEFINE macro-instruction. If the logical operator AND is specified, a test result is affirmative only when the conditions of both logical flags are 1. If the logical operator OR is specified, a test result is affirmative when the condition of either logical flag (or of both) is 1.

$value_1$
specifies the value of a data item. This operand can be written as the symbolic name of a program-testing counter.

relo
specifies one of the following relational operators:

LT - less than
LE - less than or equal
EQ - equal
NE - not equal
GT - greater than
GE - greater than or equal

$value_2$
specifies a second value. This operand can be written as the symbolic name of a program-testing counter. It can be written as a literal only if value 1 is not written as a literal.

A test result is affirmative when the arithmetic relationship between $value_1$ and $value_2$ is as expressed by the relational operator specified.

macro
specifies the symbolic name of the next TESTRAN macro-instruction to be encountered when a test result is affirmative. This name must not be AND or OR if the $flag_1$ operand is present and the logo and $flag_2$ operands are omitted.

DATAM
specifies the data attributes (type, length) to be used in comparing the data items specified by $value_1$ and $value_2$. If this operand is omitted, the type and length attributes defined in the symbol table for $value_1$ or $value_2$ (in that sequence) are used. If both $value_1$ and $value_2$ are specified by external symbols, a length of one byte is assumed. This operand is ignored when flag operands are used or when names of program-testing counters are used as value operands.

Note: Scale attributes defined in the symbol table or specified by this operand are ignored.

EXAMPLE: In the following example, when ANDing the logical flags named RED and BLUE results in the condition 1, the TESTRAN macro-instruction named QUIT is the next to be encountered. If the resulting condition is 0, the macro-instruction named NEXT is encountered. If the data item at the address specified by TABLE plus contents of index register 4 is greater than the data item at MAXIMUM, the macro-instruction named QUIT is again the next to be encountered.

```
        TEST  WHEN,RED,AND,BLUE,QUIT
NEXT    TEST  WHEN,TABLE(4),GT,MAXIMUM,QUIT
        .     .
        .     .
        .     .
```

TEST ON -- Alter Test Sequence on Counter Interval

This form of the TEST macro-instruction controls the sequence in which other test services are performed by incrementing a counter and then testing it for specified values each time the macro-instruction is encountered at execution time. An affirmative test result causes the TESTRAN interpreter to perform the services specified by a series of TESTRAN macro-instructions, the first of which is specified by its symbolic name. A nonaffirmative test result causes the TESTRAN interpreter to perform the test services specified by the series of macro-instructions that begins with the next sequential macro-instruction.

Name	Operation	Operand
[symbol]	TEST	ON,[low-{integer / adval[1]}],[high-{integer / adval[1]}] ,[interval-{integer / adval[1]}],macro-symbol [,COUNTER=symbol]

[1]This operand cannot be written as a literal.

ON
specifies the TEST ON form of the TEST macro-instruction.

low
specifies the lowest value in the range of values for which the counter is tested; if this operand is omitted, the lowest value in the range is assumed to be 1.

high
specifies the highest value in the range of values for which the counter is tested; if this operand is omitted, the highest value in the range is assumed to be the maximum value of the counter ($2^{31}-1$).

interval
specifies the counter intervals at which affirmative tests occur; if this operand is omitted, the value of the interval is assumed to be 1.

An affirmative test result occurs when the value of the counter is an integral multiple of the interval and falls within the range defined by the low and high operands. Values specified for the low, high, and interval operands must be integers in the range of 1 to $2^{31}-1$.

macro
specifies the symbolic name of the next TESTRAN macro-instruction to be encountered when a test result is affirmative.

COUNTER
specifies a program-testing counter defined by a TEST DEFINE macro-instruction. This counter can be preset to any value from 0 to $2^{31}-1$ by a SET COUNTER macro-instruction; it is incremented and tested by all TEST ON macro-instructions in which it is specified. If this operand is omitted, an unnamed counter with an initial value of 0 is defined by the TESTRAN interpreter; this counter cannot be used by other TEST ON macro-instructions. Whether named or unnamed, the counter is incremented by one each time the macro-instruction is encountered.

EXAMPLE: In the following example, EX1 increments the value of LOOPCNT by one; it causes the series of macro-instructions beginning at TROUBLE to be encountered whenever the value of LOOPCNT is 8, 12, or 16.

```
EX1      TEST  ON,7,16,4,TROUBLE,COUNTER=LOOPCNT
```

TEST CLOSE -- Terminate Testing

This form of the TEST macro-instruction terminates program testing initiated by an associated TEST OPEN macro-instruction. The TESTRAN interpreter then executes the instruction displaced by the last executed TESTRAN SVC and returns control to the problem program. For a detailed description of the use of this macro-instruction, refer to "Notes on Usage."

Name	Operation	Operand
[symbol]	TEST	CLOSE

CLOSE
specifies the TEST CLOSE form of the TEST macro-instruction.

EXAMPLE: In the following example, the TEST CLOSE macro-instruction terminates program testing initiated by OP1 and OP2. Control is returned to the problem program instruction named ABC.

```
OP1       TEST  OPEN,ENTRY,OPTEST=OP2
          .
          .
          .
OP2       TEST  OPEN
          TEST  AT,ABC
          .
          .
          .
          TEST  CLOSE
```

GO TO -- Encounter TESTRAN Macro-Instruction

This form of the GO macro-instruction causes the TESTRAN interpreter to perform the sequence of test services specified by a series of TESTRAN macro-instructions, the first of which is specified by its symbolic name.

Name	Operation	Operand
[symbol]	GO	TO,macro-symbol

TO
specifies the GO TO form of the GO macro-instruction.

macro
specifies the symbolic name of the next TESTRAN macro-instruction to be encountered.

EXAMPLE: In the following example, EX1 causes program testing to continue with the TESTRAN macro-instruction whose symbolic name is CHECK.

```
EX1       GO    TO,CHECK
```

PROGRAMMING NOTES: TEST AT, TEST OPEN, and TEST DEFINE macro-instructions should not be specified by the macro operand in this form of the GO macro-instruction. If they are specified in this way, TEST AT is interpreted as a GO BACK with no return operand, and TEST OPEN and TEST DEFINE are ignored.

Overlay segments are not automatically loaded when the specified macro-instruction is in a segment not currently in storage. Instead, an error message is generated and the macro-instruction is ignored.

GO IN -- Enter TESTRAN Subroutine

This form of the GO macro-instruction causes the TESTRAN interpreter to perform the sequence of test services specified by a series of TESTRAN macro-instructions, the first of which is specified by its symbolic name. The address of the TESTRAN macro-instruction following the GO IN is saved by the TESTRAN interpreter, enabling the GO IN and GO OUT macro-instructions to be used in combination to provide a subroutine capability.

Name	Operation	Operand
[symbol]	GO	IN,macro-symbol

IN
specifies the GO IN form of the GO macro-instruction.

macro
specifies the symbolic name of the next TESTRAN macro-instruction to be encountered.

EXAMPLE: In the following example, EX1 causes program testing to continue with the macro-instruction named DISPLAY. The address of the macro-instruction that follows EX1 is saved.

```
EX1      GO    IN,DISPLAY
```

PROGRAMMING NOTES: Refer to the programming notes for GO TO.

GO OUT -- Return from TESTRAN Subroutine

This form of the GO macro-instruction causes the TESTRAN interpreter to terminate the sequence of test services performed as a result of a previously encountered GO IN macro-instruction. The address of the next TESTRAN macro-instruction to be encountered is that saved by the associated GO IN macro-instruction. The TESTRAN interpreter can maintain a maximum of three return addresses, making possible a maximum of three levels of subroutines. If more than three levels of subroutines are created, only the three most recent return addresses will be saved. If no GO IN macro-instruction has been encountered, the GO OUT is interpreted as a GO BACK.

Name	Operation	Operand
[symbol]	GO	OUT

OUT
specifies the GO OUT form of the GO macro-instruction.

EXAMPLE: The test macro-instructions shown in the following example are encountered at execution time in the order expressed by the digits included in the symbolic names.

The GO IN macro-instruction named XXXX1 saves the address of the macro-instruction named XXXX7 and causes the series of macro-instructions beginning at YY2 to be encountered.

The series of macro-instructions beginning at YY2 includes the GO IN macro-instruction named YY3. YY3 saves the address of the macro-instruction named YY6 and causes the series of macro-instructions beginning at Z4 to be encountered.

The series of macro-instructions beginning at Z4 includes the GO OUT macro-instruction named Z5. Z5 terminates the series and causes the macro-instruction named YY6 to be encountered.

The GO OUT macro-instruction named YY6 terminates the series of macro-instructions that began at YY2 and causes the macro-instruction named XXXX7 to be encountered.

```
XXXX1     GO    IN,YY2
XXXX7     TEST  CLOSE
*
YY2       DUMP  PANEL                  ***
          .     .                        *
          .     .                        * SUBROUTINE YY2
          .     .                        *
YY3       GO    IN,Z4                    *
YY6       GO    OUT                    ***
*
Z4        DUMP  MAP                    ***
          .     .                        *
          .     .                        * SUBROUTINE Z4
          .     .                        *
Z5        GO    OUT                    ***
```

GO BACK -- Return to Problem Program

This form of the GO macro-instruction causes the TESTRAN interpreter to return control to the problem program.

Name	Operation	Operand
[symbol]	GO	BACK[,return-addx]

BACK
specifies the GO BACK form of the GO macro-instruction.

return
specifies an address in the problem program to which control is to be returned by the TESTRAN interpreter. If this operand is omitted, the TESTRAN interpreter executes the instruction displaced by the last executed TESTRAN SVC and returns control to the next sequential instruction in the problem program. (The displaced instruction is not executed if the return operand is present.)

EXAMPLE: The GO BACK macro-instruction in the following example returns control to a specific problem program address (GETNEXT) rather than to the instruction specified by the related TEST AT macro-instruction. This alteration of the normal problem program control flow and the reason for it are recorded by the DUMP COMMENT macro-instruction.

```
DUMP  COMMENT,'ERROR, NETPAY IS NEGATIVE.  CONTROL RETURNED
      TO GETNEXT',SELECT=1
GO    BACK,GETNEXT
```

PROGRAMMING NOTES: The optional return operand of the GO BACK macro-instruction enables the programmer to alter the normal control flow of his problem program. This alteration will affect the processing performed by the program and may cause sections of the program that contain errors to be left unexecuted. The operand is useful when an error condition (such as an endless loop) has been detected, and the programmer wishes to bypass the error and to continue testing at some other point in the program. If a GO BACK macro-instruction that includes this operand is encountered, the problem program must not be expected to function in the same manner when run independently as when run under TESTRAN supervision. The programmer should therefore design his TESTRAN macro-instructions to cause an indication of any such GO BACK encountered, e.g., by use of a preceding DUMP COMMENT macro-instruction.

SET FLAG -- Assign Condition to Flag

This form of the SET macro-instruction assigns a condition to a logical flag defined by a TEST DEFINE macro-instruction.

Name	Operation	Operand
[symbol]	SET	FLAG,$flag_1$-symbol,{$flag_2$-symbol | condition-{=0|=1}}

FLAG
specifies the SET FLAG form of the SET macro-instruction.

$flag_1$
specifies the symbolic name of the flag to which the condition is to be assigned.

$flag_2$
specifies the symbolic name of a flag that has the condition to be assigned to $flag_1$.

condition
specifies the condition (either 0 or 1) to be assigned to $flag_1$.

EXAMPLE: In the following example, EX1 assigns a condition of 1 to the logical flag whose symbolic name is BLUE.

```
EX1       SET    FLAG,BLUE,=1
```

SET COUNTER -- Assign Value to Counter

This form of the SET macro-instruction assigns a value to a program-testing counter defined by a TEST DEFINE macro-instruction.

Name	Operation	Operand
[symbol]	SET	COUNTER,ctr-symbol,set-adval

COUNTER
specifies the SET COUNTER form of the SET macro-instruction.

ctr
specifies the symbolic name of the counter to which the value is assigned.

set
specifies the value assigned to the counter; the value may be any integer in the range from -2^{31} to $+2^{31}-1$.

EXAMPLE: In the following example, EX1 assigns the value contained in general register 9 to the counter named LOOPCNT.

```
EX1       SET    COUNTER,LOOPCNT,G'9'
```

SET VARIABLE -- Assign Value to Storage or Register

This form of the SET macro-instruction assigns a value to a specified register or data item in the problem program.

Name	Operation	Operand
[symbol]	SET	VARIABLE,vari-adval[1],set-adval[,DATAM=tls]
[1] This operand cannot be written as a literal.		

VARIABLE
specifies the SET VARIABLE form of the SET macro-instruction.

vari
specifies a variable that is the contents of either a register or a location in main storage. If specified by an address, the variable must have the storage protection key of the current task.

set
specifies the value assigned. This operand can be written as the symbolic name of a program-testing counter that has the value to be assigned to the variable.

DATAM
specifies the data attributes (type, length) to be used by the TESTRAN interpreter in assigning the value to the variable. If this operand is omitted, the symbol table attributes for the variable or the value (in that sequence) are used. If both variable and value are specified by external symbols, a length of one byte is assumed.

Note: The type attribute can be used to imply a length attribute as described in the discussion of data attribute notation in Appendix A. The type attribute has no other function for this macro-instruction. Scale attributes defined in the symbol table or specified by this operand are ignored.

EXAMPLE: In the following example, EX1 assigns the value of the 64 bytes beginning at ASSUMED to the 64 bytes beginning at UNKNOWN.

```
EX1      SET   VARIABLE,UNKNOWN,ASSUMED,DATAM=L64
```

PROGRAMMING NOTES: The SET VARIABLE macro-instruction enables the programmer to alter the values of data within his problem program. This alteration may affect other data or control flow or both. This form of the macro-instruction is useful when an error condition (such as an unreasonable result of a computation) has been detected, and the programmer wishes to assume some standard value and continue testing. If a SET VARIABLE macro-instruction is encountered, the problem program must not be expected to function in the same manner when run independently as when run under TESTRAN supervision. The programmer should therefore design his TESTRAN macro-instructions to cause an indication of any SET VARIABLE encountered, e.g., by use of a preceding DUMP COMMENT macro-instruction.

NOTES ON USAGE

This section contains notes on the usage of TESTRAN macro-instructions and their operands. These notes present detailed information essential to proper usage of the TESTRAN facility but not necessary for basic understanding of the services provided.

Keyword Modifiers

Keyword modifiers are optional keyword operands that have standard formats and can be used in more than one macro-instruction. Each type of keyword modifier is described in the following paragraphs.

Output Selection Modifier (SELECT Operand): The output selection modifier causes a specified output selection code to be associated with test data produced by macro-instructions to which the modifier applies.

- If specified in a TEST OPEN macro-instruction, the modifier applies to all action macro-instructions that follow the TEST OPEN (and precede any subsequent TEST OPEN) in the source program. The modifier overrides any output selection codes specified in TEST OPEN macro-instructions specified in the OPTEST operand.

- If specified in a TEST AT macro-instruction, the modifier applies to all action macro-instructions that follow the TEST AT (and precede any subsequent TEST AT or TEST OPEN) in the source program. For those macro-instructions to which it applies, the modifier overrides any output selection code specified in a related TEST OPEN macro-instruction.

- If specified in an action macro-instruction, the modifier applies to the macro-instruction in which it appears. For this macro-instruction, the modifier overrides any output selection code specified in a related TEST OPEN or TEST AT macro-instruction.

To select data for editing and printing, the associated output selection codes are specified in a job control statement for the TESTRAN editor. A blank is used in the job control statement to select test data produced by action macro-instructions to which no output selection modifiers apply.

Data resulting from execution of an asynchronous exit routine may be masked by surrounding data if produced during an interruption of the TESTRAN interpreter. The asynchronous data can be selected by its output selection code, but only if this code differs from that of the surrounding data, and only if the surrounding data is not selected during the same job step. Macro-instructions encountered as a result of an asynchronous interruption should, therefore, specify output selection codes different from those specified by other macro-instructions encountered under the same task.

Careful use of output selection modifiers enables the programmer to identify test data relevant to particular program areas and to limit printed output accordingly.

Data Modifier (DATAM Operand): The data modifier causes a specified set of data attributes to be associated with data that is recorded, compared, or modified by the macro-instruction in which the modifier appears. For a full description of the way in which attributes are specified, refer to the discussion of data attribute notation in Appendix A.

Data attributes are used both during assembly and during output editing. During assembly, data attributes determine the type and length of data to be recorded, compared, or modified by the following macro-instructions:

DUMP DATA DUMP CHANGES TRACE REFER	} -	when the end operand is omitted
TEST WHEN	-	when an arithmetic relationship between data in main storage is specified
SET VARIABLE		

The scale attribute (if any) is ignored. If no modifier is present, the attributes used are those corresponding to the first address that is specified as an operand. These attributes are those defined in the

symbol table for the first symbol included in the address. If the symbol is an external symbol, the attributes for the second address operand (if any) are used. If no attributes can be determined in this manner, the type is assumed to be hexadecimal and the length is assumed to be 1 byte.

During output editing, data attributes determine the type, length, and scale of data that is to be printed. If a data modifier is present, the length attribute determines the length of each field that is printed, and the type and scale attributes determine the format of each field. If no modifier is present, the symbol table determines the division of data into fields and the printing format of complete fields. Partial fields, and fields not presented in the symbol table, are printed in 4-byte hexadecimal format.[1]

Use of data modifiers enables the programmer to override data attributes defined in the symbol table, and to specify attributes when no symbol table exists. A data modifier should always be used when address arithmetic, external references, or absence of a symbol table would otherwise cause an incorrect assumption of data attributes.

Name Modifier (NAME Operand): The name modifier causes a specified symbolic name to be associated with test data produced by the macro-instruction in which the modifier appears.

The symbolic name specified is printed with the edited test data; printing of other symbolic names associated with the same data is suppressed. If the name modifier is omitted, test data is printed with the symbolic names defined in the source program symbol table.

Use of name modifiers enables the programmer to label printed output of test data when no symbol table is available or when the test data is not identified by a symbolic name in the source program.

Comment Modifier (COMMENT Operand): The comment modifier causes a specified comment to be associated with the macro-instruction in which the modifier appears. The specified comment is printed with the associated test data. The maximum length is 120 characters.

Use of comment modifiers enables the programmer to annotate traces of transfers, subroutine calls, and references to data.

Dummy Section Modifier (DSECT Operand): The dummy section modifier indicates that addresses used in previous operands of the same macro-instruction are associated with a dummy control section.

Use of dummy section modifiers enables the programmer to refer to data in dynamically allocated storage and to print it in the format defined for an associated dummy control section.

---

[1]The assumption of a 4-byte hexadecimal format may fail in the case of data whose location was previously occupied by a load module executed under the current task. If the load module contained a symbol table, the attributes defined in the symbol table are used, even though the load module is no longer in storage. Therefore, to ensure use of appropriate attributes, the programmer should use a data modifier (or a dummy section modifier) when referring to data in an allocated storage area or in a load module that does not include a symbol table.

Address Specification

When a TESTRAN macro-instruction is encountered, the locations specified by its operands must be (1) included in the same load module as the encountered macro-instruction, and (2), in the case of planned overlay, included in the same overlay segment as the encountered macro-instruction. Addresses that refer to locations in other object modules must be listed in EXTRN and ENTRY statements according to the rules for external references and entry points. These include addresses specified by:

- Address operands referring to the problem program.
- Operands that designate system tables.
- Names of other TESTRAN macro-instructions.
- Names of program-testing counters and logical flags.

Two exceptions to these rules occur in the case of the TEST OPEN macro-instruction:

1. Because the symbolic name of each TEST OPEN macro-instruction is also the name of a control section, ENTRY statements are not required to identify TEST OPEN macro-instructions as object module entry points.

2. In the case of planned overlay, the OPTEST operand can specify the symbolic names of TEST OPEN macro-instructions that are located in segments not currently in main storage.

TEST OPEN Macro-Instructions

A TEST OPEN macro-instruction must be the first TESTRAN macro-instruction encountered during assembly. TESTRAN macro-instructions that precede the first TEST OPEN are ignored, and are identified by diagnostic messages in the assembly listing.

An unlimited number of TEST OPEN macro-instructions can be processed during a single assembly. The symbolic name of each is assigned to a control section that contains the macro-expansions corresponding to a series of TESTRAN macro-instructions. This series includes the TEST OPEN and all TESTRAN macro-instructions that follow the TEST OPEN (and precede any subsequent TEST OPEN) in the source module.

The first TEST OPEN macro-instruction executed during each task determines the limits on test execution and output for the duration of the task. These items are ignored if specified in other TEST OPEN macro-instructions executed as part of the same task.

Execution-time testing is initiated only when control is passed to a TEST OPEN macro-instruction. Execution of this macro-instruction opens the control section of which it is a part and also any other control sections designated by the OPTEST operand; all control sections opened by the same TEST OPEN must be included in the same load module.

Opening of each control section causes TESTRAN SVC's to be inserted at problem program addresses specified by TEST AT macro-instructions in the opened control section. Opening also causes acquisition of the main storage required for TESTRAN internal tables. A maximum of 255 such openings can occur during execution of a task. Attempts to reopen a control section that is currently open are ignored.

Except when a load module is an overlay structure, each control section can be opened independently by execution of the appropriate TEST OPEN macro-instruction. In the case of planned overlay, only a single TEST OPEN macro-instruction can be executed successfully. This macro-instruction must be located in the root segment; when executed, it opens the control section of which it is a part and causes opening of any control sections designated by the OPTEST operand. Designated control sections not currently in main storage are opened automatically upon loading of the segments in which they are located. When a segment is overlaid, control sections contained within it are temporarily closed; these control sections are automatically reopened when the segment is reloaded.

TEST CLOSE Macro-Instructions

Testing is suspended when a TEST CLOSE macro-instruction is encountered in the sequence of TESTRAN macro-instructions. This macro-instruction closes the control section of which it is a part and also any other control sections that were opened by the same TEST OPEN macro-instruction. Closing of each control section causes suspension of active traces and re-insertion of problem program instructions at the addresses specified by TEST AT macro-instructions in the closed control section. It also causes the release of main storage that was dynamically acquired for TESTRAN internal tables.

Control sections located in programs that receive control through LINK or XCTL macro-instructions should always be closed before control is passed or returned to some other program. Closing of these control sections prevents unnecessary duplication of internal tables when such programs are repeatedly loaded and executed. The closed control sections can later be reopened by execution of the appropriate TEST OPEN macro-instructions.

Editing Restrictions

Input to the TESTRAN editor includes all test data recorded during execution of a task, plus certain control data produced by the TESTRAN interpreter. The control data is entered into internal tables and is used to control the editing process. The internal tables are limited in size, and this limitation places restrictions on source program design and on output editing. These restrictions are discussed in the following paragraphs.

Program Sectioning: All programs executed under a single task should consist collectively of no more than 50 control sections, dummy sections, and blank common sections. This limit includes control sections containing TESTRAN macro-instructions, and each section defined in a program is counted once for each area of storage into which the program is loaded. If this limit is exceeded, an error message will be printed, and data from the excess control sections will be printed in 4-byte hexadecimal format.

TESTRAN Openings: No more than 50 control sections containing TESTRAN macro-instructions should be opened during execution of a single task. Each control section is counted once for each time it is opened. If this limit is exceeded, data produced by macro-instructions in the excess control sections will be ignored.

Inserted SVC Instructions: No more than 100 TESTRAN SVC instructions should be inserted in a problem program during execution of a single task. This limit includes each problem program address specified in a TEST AT macro-instruction, counting each address once for each opening of the control section. If this limit is exceeded, an error message will be printed, and instructions displaced by the excess SVC instructions will not appear in dumps of the problem program.

Change Dumps: No more than 40 change dumps (the output of 40 DUMP CHANGES macro-instructions) should be selected for editing during a single job step. Each DUMP CHANGES macro-instruction is counted once for each opening of its control section that results in the macro-instruction being encountered. If this limit is exceeded, an error message will be printed, and all additional change dumps will be ignored.

Note: Editing restrictions are more severe if less than the normal amount of main storage (17K bytes) is available to the TESTRAN editor. The limit on program sectioning is reduced to 25 sections of all types. TESTRAN openings are limited to 10, inserted SVC instructions to 20, and change dumps to 10.

Improperly Coded Macro-Instructions

Assembly is never terminated because of an improperly coded TESTRAN macro-instruction; instead, the assembler takes appropriate corrective action and inserts a diagnostic message in the assembly listing. There are three levels of error severity:

- Severity 4. The macro-instruction is expanded, but the invalid operand is ignored; or, the macro-instruction is expanded, the error is ignored, and a standard case is assumed.
- Severity 8. The macro-instruction is not expanded.
- Severity 12. The macro-instruction is not expanded. Subsequent TESTRAN macro-instructions, preceding the next valid TEST OPEN macro-instruction, are ignored.

Unless corrected by the programmer, coding errors may cause errors during execution of TESTRAN service routines. Diagnostic error messages are inserted in the test output when such errors occur during execution.

EDITED OUTPUT FORMATS

Edited test data is printed on the system output device in a column 120 characters wide. Each page of the output includes a standard page heading and an average of 55 lines of test data produced by one or more TESTRAN macro-instructions. The types of lines that can appear are as follows:

- Standard page heading.
- Dump output lines.
  DUMP DATA, DUMP CHANGES
  DUMP MAP
  DUMP TABLE
  DUMP PANEL
  DUMP COMMENT

- Trace output lines.
  Initial trace output lines
  TRACE FLOW
  TRACE CALL
  TRACE REFER
  TRACE STOP
- Output lines for control macro-instructions.
  TEST OPEN
  TEST AT
  TEST CLOSE
  Other encountered control macro-instructions
- Error message lines.

The printing formats for specific data types are listed in Table 35.

Table 35. Printing Formats for Data Types

Data Type	Assumed Length in Bytes (1)	Printing Format (2)
Character (3)	1	C
Hexadecimal	1	XX
Binary	1	BBBBBBBB
Fixed-point (half-word)	2	SDDDDD (4)
Fixed-point (full-word)	4	SDDDDDDDDD (4)
Short floating-point	4	S0.DDDDDDDD ESDD
Long floating-point	8	S0.DDDDDDDDDDDDDDD ESDD
Packed decimal	1	SD
Zoned decimal	1	SD
Address (5)		
Instruction:		
RR format	2	CODE XX
RS, RX, and SI formats	4	CODE XX X XXX
SS format	6	CODE XX X XXX X XXX

Notes to Table 35:

1. The lengths assumed in definitions of printing formats are the assembler implied lengths for the corresponding data types. (Refer to Appendix A, Table 38.)

2. The letters shown in definitions of printing formats have the following meanings:

   C is one EBCDIC character.
   X is one hexadecimal digit.
   B is one binary digit.

S is an algebraic sign (+ or -).
D is one decimal digit.
0 is a high order zero.
E means 'exponent'; the succeeding signed pair of digits is the exponent of the floating-point number.
CODE is a machine mnemonic operation code.

3. Unprintable characters (other than blanks) are printed as two hexadecimal digits, the second of which appears on a separate line immediately below the first. For example, the hexadecimal data

   C1D3D7C8C103C4C1E3C1

   when edited into character format, is printed as

   ALPHA0DATA
        3

4. This format includes a decimal point that is positioned according to the scale factor associated with the data.

5. All addresses are printed in their source language formats.

Standard Page Heading

The standard page heading for printed test output is as follows:

IIIIIIII
is the output identification (the ident operand, if any, of the first-executed TEST OPEN macro-instruction).

DD/DDD
is the current date (year/day).

TT/TT
is the time (hour/minute) at which editing was begun.

NNNN
is the output page number.

Output Lines for DUMP DATA and DUMP CHANGES

The output lines for DUMP DATA and DUMP CHANGES include subheadings that identify each control section for which data is printed. Recorded test data is represented by one or more paired lines of print, each containing a variable number of data entries.

The format of the output lines is as follows:

```
P) MACRO ID NNN, DUMP CCCCCCC STARTING IN SECTION CSCSCSCS
  AAAA     S1S1S1S1      S2S2S2S2      S3S3S3S3
LLLLLL     D1D1D1D1D1D1D1 D2D2D2D2D2D2D2 D3D3D3D3D3D3D3
```

P
is the output selection code (if any) associated with the test data produced by the macro-instruction.

NNN
is the macro-instruction identification number assigned by the assembler.

CCCCCCC
is the operand DATA or CHANGES.

CSCSCSCS
is the symbolic name (if any) of the control section that contains the displayed data.

AAAA
is the assembled address of the first data entry.

LLLLLL
is the loaded address of the first data entry.

S1S1S1S1
is the symbolic name (if any) of the first data entry.

D1D1D1D1D1D1D1
is the first data entry.

S2S2S2S2
is the symbolic name (if any) of the second data entry.

D2D2D2D2D2D2D2
is the second data entry.

S3S3S3S3
is the symbolic name (if any) of the third data entry.

D3D3D3D3D3D3D3
is the third data entry.

NOTE: The number of named data entries per line varies from 1 to 11 due to differences in length; starting positions are a minimum of 9 printing positions apart. Data entries too long for the current line are started on a new line.

If the displayed data includes an inserted TESTRAN SVC, the displaced data (i.e., instruction) is printed in the data entry. The inserted SVC is printed immediately below the original instruction.

Output Lines for DUMP MAP

The output lines for DUMP MAP include one line for each control section associated with the task current when the DUMP MAP macro-instruction was encountered. Preceding these lines is a line of column headings that identifies the information printed for each control section.

The format of the output lines is as follows:

```
P) MACRO ID NNN, DUMP MAP
        NAME          TYPE          CSECT NAME   ASSEMBLED AT   LOADED AT   LENGTH
      N1N1N1N1        TTTT          CSCS1111       AAAAAA         LLLLLL      LNLN
```

P
: is the output selection code (if any) associated with the test data produced by the macro-instruction.

NNN
: is the macro-instruction identification number assigned by the assembler.

N1N1N1N1
: is the symbolic name of a program associated with the current task; this name is printed only when different from that which applies to the previous line.

TTTT
: is the words LOADED PROGRAM or OBTAINED STORAGE. LOADED PROGRAM indicates a control section for which storage was reserved at assembly time; OBTAINED STORAGE indicates a dynamically allocated storage area.

CSCS1111
: is the symbolic name (if any) of a control section associated with the task.

AAAAAA
: is the assembled address of the control section or data area.

LLLLLL
: is the loaded address of the control section or data area.

LNLN
: is the length (in base 10) of the control section or data area.

NOTE: Some of the areas included in this output will be areas allocated for use by the operating system.

Output Lines for DUMP TABLE

The output lines for DUMP TABLE include lines that identify each section of the table, and lines that display the contents of fields within each section. Preceding these lines is a line of column headings that identifies the information printed for each section and field.

The format of the output lines is as follows:

```
P) MACRO ID NNN, DUMP TABLE   NMNMNMNM LOADED AT SSSSSSSS(CSCSCSCS) AAAAAA LLLLLL
   SECTION    FIELD NAME   CONTENTS
   HDHDHDHD
              FNFNFNFN   CCCCCCCC
```

P
is the output selection code (if any) associated with the test data produced by the macro-instruction.

NNN
is the macro-instruction identification number assigned by the assembler.

NMNMNMNM
is the operand DCB, DEB, or TCB, indicating the type of table.

SSSSSSSS
is the symbolic name (if any) of the table.

CSCSCSCS
is the symbolic name (if any) of the control section that contains the table.

AAAAAA
is the assembled address (if any) of the table.

LLLLLL
is the loaded address of the table.

HDHDHDHD
is a heading that identifies a section of the table.

FNFNFNFN
is the name (if any) of a field in the identified section of the table.

CCCCCCCC
is the contents of a field.

NOTE: The name of each field begins with DCB, DEB, or TCB, according to the type of table. If a field has no name, its contents are printed in hexadecimal format.

Output Lines for DUMP PANEL

The format of the output lines is as follows:

```
P) MACRO ID NNN, DUMP PANEL
G'00' HHHHHHHH G'01' HHHHHHHH G'02' HHHHHHHH G'03' HHHHHHHH G'04' HHHHHHHH G'05' HHHHHHHH G'06' HHHHHHHH G'07' HHHHHHHH
G'08' HHHHHHHH G'09' HHHHHHHH G'10' HHHHHHHH G'11' HHHHHHHH G'12' HHHHHHHH G'13' HHHHHHHH G'14' HHHHHHHH G'15' HHHHHHHH
PSW  HH H H HHHH H H HHHHHH  CC=D  FIX POINT OVERFLOW FFF  DEC OVERFLOW FFF  EXP UNDERFLOW FFF  SIGNIFICANCE FFF
  F'0' HHHHHHHH HHHHHHHH F'2' HHHHHHHH HHHHHHHH F'4' HHHHHHHH HHHHHHHH F'6' HHHHHHHH HHHHHHHH
```

P
is the output selection code (if any) associated with the test data produced by the macro-instruction.

NNN
is the macro-instruction identification number assigned by the assembler.

G
means 'general register'; the succeeding pair of digits is the number of a general register.

HHHHHHHH
is the contents of a general register in hexadecimal format.

HH H H HHHH H H HHHHHH
is the complete PSW in hexadecimal format.

CC
means 'condition code.'

D
is the decimal value of the condition code.

FFF
is the word ON or OFF.

F
means 'floating-point register'; the succeeding digit is the number of a floating-point register.

HHHHHHHH HHHHHHHH
is the contents of a floating-point register in hexadecimal format.

Output Lines for DUMP COMMENT

The format of the output lines is as follows:

```
P) MACRO ID NNN, DUMP COMMENT
CCCCC
```

P
is the output selection code (if any) associated with the comment.

NNN
is the macro-instruction identification number assigned by the assembler.

CCCCC
is the recorded comment; the maximum length is 120 characters.

Initial Trace Output Lines

The initial trace output lines indicate the start of a new trace when a TRACE FLOW, TRACE CALL, or TRACE REFER macro-instruction has been encountered.

The format of the output lines is as follows:

```
P) MACRO ID NNN, TRACE TTTT , TTTTTTTT, FROM SSSS1111(CSCS1111) AAA111 LLL111 TO SSSS2222(CSCS2222) AAA222 LLL222,
STARTED
CCCCC
```

P
is the output selection code (if any) associated with the test data produced by the macro-instruction.

NNN
is the macro-instruction identification number assigned by the assembler.

TTTT
is the operand FLOW, CALL, or REFER.

TTTTTTTT
is the symbolic name of the TESTRAN control section that contains the TRACE macro-instruction.

SSSS1111
is the symbolic name (if any) of the starting location of the traced storage area.

CSCS1111
is the symbolic name (if any) of the control section that contains the starting location.

AAA111
is the assembled starting address.

LLL111
is the loaded starting address.

SSSS2222
is the symbolic name (if any) of the ending location of the traced storage area.

CSCS2222
is the symbolic name (if any) of the control section that contains the traced storage area.

AAA222
is the assembled ending address.

LLL222
is the loaded ending address.

CCCCC
is the recorded comment; the maximum length is 120 characters.

Output Lines for TRACE FLOW

The output lines for TRACE FLOW indicate the occurrence of a program transfer. Other output lines, indicating the start of a trace, are described in "Initial Trace Output Lines." The two sets of output lines may appear together or may be separated by other test data.

There are two printing formats for TRACE FLOW output lines: the normal format and the special format. The special format is used when the program transfer resulted from execution of an EX instruction. Both are shown below.

The normal format is as follows:

```
P) MACRO ID NNN, TRACE FLOW  , TTTTTTTT, FROM SSSS1111(CSCS1111) AAA111 LLL111 TO SSSS2222(CSCS2222) AAA222 LLL222, CC=D
IIIIIIIIIIIIIIIIII  G'N1' XXXX1111 G'N2' XXXX2222
CCCCC
```

P
: is the output selection code (if any) associated with the test data produced by the macro-instruction.

NNN
: is the macro-instruction identification number assigned by the assembler.

TTTTTTTT
: is the symbolic name of the TESTRAN control section that contains the TRACE FLOW macro-instruction.

SSSS1111
: is the symbolic name (if any) of the location from which the program transfer was made.

CSCS1111
: is the symbolic name (if any) of the control section from which the program transfer was made.

AAA111
: is the assembled address of the program transfer (branch or SVC) instruction.

LLL111
: is the loaded address of the program transfer instruction.

SSSS2222
: is the symbolic name (if any) of the location to which the program transfer was made.

CSCS2222
: is the symbolic name (if any) of the control section to which the program transfer was made.

AAA222
: is the assembled address of the location to which the program transfer was made.

LLL222
: is the loaded address of the location to which the program transfer was made.

CC
means 'condition code.'

D
is the decimal value of the condition code.

IIIIIIIIIIIIIIIII
is the program transfer instruction; its format is that defined for instructions in Table 35.

G
means 'general register.'

N1
is the number of the first general register used to form the effective address of the instruction.

XXXX1111
is the contents of general register N1 in hexadecimal format.

N2
is the number of the second general register used to form the effective address of the instruction.

XXXX2222
is the contents of general register N2 in hexadecimal format.

CCCCC
is the recorded comment (if any); the maximum length is 120 characters.

The special format is as follows:

```
P) MACRO ID NNN, TRACE FLOW  , TTTTTTTT, FROM SSSS1111(CSCS1111) AAA111 LLL111 TO SSSS2222(CSCS2222) AAA222 LLL222, CC=D
I1I1I1I1I1I1I1I1I1I1 EXECUTED AS I2I2I2I2I2I2I2I2I2I2 BY I3I3I3I3I3I3I3I3 FROM LOCATION SSSSSSSS(CSCSCSCS) AAAAAA LLLLLL
G'NN' XXXXXXXX G'NN' XXXXXXXX G'NN' XXXXXXXX G'NN' XXXXXXXX G'NN' XXXXXXXX G'NN' XXXXXXXX G'NN' XXXXXXXX
CCCCC
```

P
is the output selection code (if any) associated with the test data produced by the macro-instruction.

NNN
is the macro-instruction identification number assigned by the assembler.

TTTTTTTT
is the symbolic name of the TESTRAN control section that contains the TRACE FLOW macro-instruction.

SSSS1111
is the symbolic name (if any) of the program transfer (branch or SVC instruction).

CSCS1111
is the symbolic name (if any) of the control section that contains the program transfer instruction.

AAA111
is the assembled address of the program transfer instruction.

LLL111
is the loaded address of the program transfer instruction.

SSSS2222
is the symbolic name (if any) of the location to which the program transfer was made.

CSCS2222
is the symbolic name (if any) of the control section to which the program transfer was made.

AAA222
is the assembled address of the location to which the program transfer was made.

LLL222
is the loaded address of the location to which the program transfer was made.

CC
means 'condition code.'

D
is the decimal value of the condition code.

I1I1I1I1I1I1I1I1I1I1I
is the program transfer instruction as it appeared in storage.

I2I2I2I2I2I2I2I2I2I2I
is the program transfer instruction as it was executed.

I3I3I3I3I3I3I
is the EX instruction.

SSSSSSSS
is the symbolic name (if any) of the EX instruction.

CSCSCSCS
is the symbolic name (if any) of the control section that contains the EX instruction.

AAAAAA
is the assembled address of the EX instruction.

LLLLLL
is the loaded address of the EX instruction.

G
means 'general register.'

NN
is the number of the general register.

XXXXXXXX
is the contents of the general register in hexadecimal format.

CCCCC
is the recorded comment (if any); the maximum length is 120 characters.

NOTE: The contents of all registers used by the two instructions are displayed. The instructions are displayed in the format defined in Table 35.

Output Lines for TRACE CALL

The output lines for TRACE CALL indicate the execution of a CALL macro-instruction. Other output lines, indicating the start of the trace, are described in "Initial Trace Output Lines." The two sets of output lines may appear together or may be separated by other test data.

The format of the output lines for TRACE CALL is as follows:

```
P) MACRO ID NNN, TRACE CALL  , TTTTTTTT, TO SSSSSSSS(CSCSCSCS) AAAAAA LLLLLL AT SSSS1111(CSCS1111) AAA111 LLL111
G'00' HHHHHHHH G'01' HHHHHHHH G'02' HHHHHHHH G'03' HHHHHHHH G'04' HHHHHHHH G'05' HHHHHHHH G'06' HHHHHHHH G'07' HHHHHHHH
G'08' HHHHHHHH G'09' HHHHHHHH G'10' HHHHHHHH G'11' HHHHHHHH G'12' HHHHHHHH G'13' HHHHHHHH G'14' HHHHHHHH G'15' HHHHHHHH
CCCCC
```

P
: is the output selection code (if any) associated with the test data produced by the macro-instruction.

NNN
: is the macro-instruction identification number assigned by the assembler.

TTTTTTTT
: is the symbolic name of the TESTRAN control section that contains the TRACE CALL macro-instruction.

SSSSSSSS
: is the symbolic name (if any) of the subroutine.

CSCSCSCS
: is the symbolic name (if any) of the control section that contains the subroutine.

AAAAAA
: is the assembled address of the subroutine.

LLLLLL
: is the loaded address of the subroutine.

SSSS1111
: is the symbolic name (if any) of the CALL macro-instruction.

CSCS1111
: is the symbolic name (if any) of the control section that contains the CALL macro-instruction.

AAA111
: is the assembled address of the CALL macro-instruction.

LLL111
: is the loaded address of the CALL macro-instruction.

G
: means 'general register.'

HHHHHHH
: is the contents of the general register in hexadecimal format. All registers used in the CALL macro-expansion are displayed.

CCCCC
is the recorded comment (if any); the maximum length is 120 characters.

Output Lines for TRACE REFER

The output lines for TRACE REFER indicate a reference to data in main storage. Other output lines, indicating the start of the trace, are described in "Initial Trace Output Lines." The two sets of output lines may appear together or may be separated by other test data.

There are two printing formats for TRACE REFER output lines: the normal format and the special format. The special format is used when the recorded reference resulted from execution of an EX instruction. Both are shown below.

The normal format is as follows:

```
P) MACRO ID NNN, TRACE REFER , TTTTTTTT, TO SSSS1111(CSCS1111) AAA111 LLL111 FROM SSSS2222(CSCS2222) AAA222 LLL222
IIIIIIIIIIIIIIIII  G'N1' XXXX1111 G'N2' XXXX2222
CCCCC
 BEFORE  CCCC1111                                  AFTER   CCCC2222
```

P
is the output selection code (if any) associated with the test data produced by the macro-instruction.

NNN
is the macro-instruction identification number assigned by the assembler.

TTTTTTTT
is the symbolic name of the TESTRAN control section that contains the TRACE REFER macro-instruction.

SSSS1111
is the symbolic name (if any) of the location to which the reference was made.

CSCS1111
is the symbolic name (if any) of the control section that contains the location to which the reference was made.

AAA111
is the assembled address of the location to which the reference was made.

LLL111
is the loaded address of the location to which the reference was made.

SSSS2222
is the symbolic name (if any) of the instruction that made the reference.

CSCS2222
is the symbolic name (if any) of the control section that contains the instruction that made the reference.

AAA222
is the assembled address of the instruction that made the reference.

LLL222
is the loaded address of the instruction that made the reference.

IIIIIIIIIIIIIIIII
is the instruction that made the reference; its format is that defined for instructions in Table 35.

G
means 'general register.'

N1
is the number of the first general register used to form the effective address of the instruction.

XXXX1111
is the contents of general register N1 in hexadecimal format.

N2
is the number of the second general register used to form the effective address of the instruction.

XXXX2222
is the contents of general register N2 in hexadecimal format.

CCCCC
is the recorded comment (if any); the maximum length is 120 characters.

CCCC1111
is the contents of the storage area before the reference to the storage area was made.

CCCC2222
is the contents of the storage area after the reference was made.

The special format is as follows:

```
P) MACRO ID NNN, TRACE REFER , TTTTTTTT, TO SSSS1111(CSCS1111) AAA111 LLL111 FROM SSSS2222(CSCS2222) AAA222 LLL222
IIIIIIIIIIIIIIIII EXECUTED AS I2I2I2I2I2I2I2I2I2I2I BY I3I3I3I3I3I3I FROM LOCATION SSSSSSSS(CSCSCSCS) AAAAAA LLLLLL
G'NN' XXXXXXXX G'NN' XXXXXXXX G'NN' XXXXXXXX G'NN' XXXXXXXX G'NN' XXXXXXXX G'NN' XXXXXXXX
CCCCC
 BEFORE  CCCC1111                                    AFTER   CCCC2222
```

P
is the output selection code (if any) associated with the test data produced by the macro-instruction.

NNN
is the macro-instruction identification number assigned by the assembler.

TTTTTTTT
is the symbolic name of the TESTRAN control section that contains the TRACE REFER macro-instruction.

SSSS1111
is the symbolic name (if any) of the location to which the reference was made.

CSCS1111
is the symbolic name (if any) of the control section that contains the location to which the reference was made.

AAA111
is the assembled address of the location to which the reference was made.

LLL111
is the loaded address of the location to which the reference was made.

SSSS2222
is the symbolic name (if any) of the instruction that made the reference.

CSCS2222
is the symbolic name (if any) of the control section that contains the instruction that made the reference.

AAA222
is the assembled address of the instruction that made the reference.

LLL222
is the loaded address of the instruction that made the reference.

I1I1I1I1I1I1I1I1I1I1I
is the instruction that made the reference as it appeared in storage.

I2I2I2I2I2I2I2I2I2I2I
is the instruction that made the reference as it was executed.

I3I3I3I3I3I3I
is the EX instruction.

SSSSSSSS
is the symbolic name (if any) of the EX instruction.

CSCSCSCS
is the symbolic name (if any) of the control section that contains the EX instruction.

AAAAAA
is the assembled address of the EX instruction.

LLLLLL
is the loaded address of the EX instruction.

G
means 'general register.'

NN
is the number of the general register.

XXXXXXXX
is the contents of the general register in hexadecimal format.

CCCCC
is the recorded comment (if any); the maximum length is 120 characters.

CCCC1111
is the contents before the reference of the storage area to which the reference was made.

CCCC2222
is the contents after the reference of the storage area to which the reference was made.

NOTE: The contents of all registers used by the two instructions are displayed. The instructions are displayed in the format defined in Table 35.

Output Line for TRACE STOP

The output line for TRACE STOP indicates suspension of one or more traces, which are identified by the macro-instruction identification numbers of the corresponding TRACE macro-instructions. Each number, or group of numbers, is preceded by the name of the control section in the macro-instruction, or group of macro-instructions, is located. If the TRACE STOP macro-instruction suspends all active traces, the word ALL is printed in place of the macro-instruction idenfification numbers.

The format of this output line is as follows:

P
is the output selection code (if any) associated with the test data produced by the macro-instruction.

NNN
is the macro-instruction identification number assigned by the assembler.

CSCSCSCS
is the symbolic name of the TESTRAN control section that contains the TRACE macro-instructions whose functions are stopped.

NN1
is the macro-instruction identification number of the first macro-instruction whose trace is stopped.

NN2
is the macro-instruction identification number of the second macro-instruction whose trace is stopped.

NN3
is the macro-instruction identification number of the third macro-instruction whose trace is stopped.

Output Lines for TEST OPEN

The output lines for TEST OPEN precede all output generated by other TESTRAN macro-instructions in the same control section (or in other control sections opened at the same time). These lines are printed for each executed TEST OPEN macro-instruction, regardless of the output selection codes used in selecting data for editing.

The format of the output lines is as follows:

```
P) MACRO ID NNN, TEST OPEN   , TESTRAN CONTROL SECTION = TTTTTTTT, IDENTIFICATION IIIIIIII
 MAXIMUM NUMBER OF PAGES MMM, MAXIMUM NUMBER OF STATEMENTS NNNN
```

P
is the output selection code (if any) specified by the SELECT operand of the TEST OPEN macro-instruction.

NNN
is the macro-instruction identification number assigned by the assembler.

TTTTTTTT
is the symbolic name of the TESTRAN control section (symbolic name of the TEST OPEN macro-instruction).

IIIIIIII
is the output identification (ident operand, if any).

MMM
is the maximum number of pages of test data produced.

NNNN
is the maximum number of TESTRAN macro-instructions encountered during execution.

Output Line for TEST AT

The output line for TEST AT indicates the execution of a TESTRAN SVC inserted in the problem program. This line is printed regardless of any output selection code specified in the TEST AT macro-instruction, but is omitted if not followed by recorded test data or an error message.

The format of this output line is as follows:

```
AT LOCATION SSSSSSSS(CSCSCSCS) AAAAAA LLLLLL ENTER TTTTTTTT
```

SSSSSSSS
: is the symbolic name (if any) of the problem program location from which the TESTRAN interpreter was entered.

CSCSCSCS
: is the symbolic name (if any) of the control section from which the TESTRAN interpreter was entered.

AAAAAA
: is the assembled address of the problem program location from which the TESTRAN interpreter was entered.

LLLLLL
: is the loaded address of the problem program location from which the TESTRAN interpreter was entered.

TTTTTTTT
: is the symbolic name of the TESTRAN control section (symbolic name of the TEST OPEN macro-instruction).

Output Line for TEST CLOSE

The output line for TEST CLOSE indicates the closing of one or more control sections consisting of TESTRAN macro-instructions. This line is printed regardless of any output selection code specified in a related TEST OPEN macro-instruction.

The format of this output line is as follows:

```
P) MACRO ID NNN, TEST CLOSE
NNNNNNNN(TTTTTTTT) AAAAAA LLLLLL
```

P
: is the output selection code (if any) specified by the SELECT operand of a related TEST OPEN macro-instruction.

NNN
is the macro-instruction identification number assigned by the assembler.

NNNNNNNN
is a symbol generated during expansion of the TEST OPEN macro-instruction.

TTTTTTTT
is the symbolic name of the TESTRAN control section (symbolic name of the related TEST OPEN macro-instruction). If more than one TESTRAN control section is closed by this macro-instruction, the symbolic name of each is displayed.

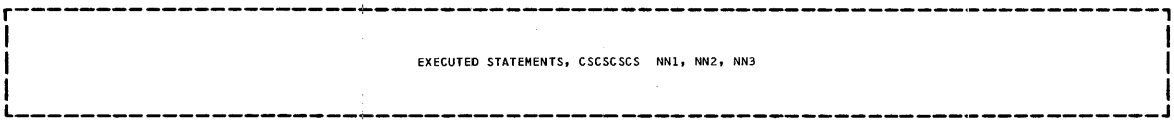
AAAAAA
is the assembled address of the TESTRAN control section (assembled address of the TEST OPEN macroinstruction).

LLLLLL
is the loaded address of the TESTRAN control section (loaded address of the TEST OPEN macro-instruction).

Output Lines for Other Encountered Control Macro-Instructions

In these output lines, macro-instruction identification numbers are listed in the order in which corresponding macro-instructions were encountered during test execution. If a macro-instruction is not in the same control section as the one previously encountered, the name of the new control section appears between the two identification numbers in the list. The list includes only the identification numbers of control macro-instructions other than TEST OPEN, TEST AT, and TEST CLOSE; it is printed only when followed by output of an action macro-instruction or by an error message.

The format of the first output line is as follows:



CSCSCSCS
is the symbolic name of the TESTRAN control section that contains the encountered TESTRAN macro-instructions.

NN1
is the macro-instruction identification number of the first control macro-instruction encountered.

NN2
is the macro-instruction identification number of the second control macro-instruction encountered.

NN3
is the macro-instruction identification number of the third control macro-instruction encountered.

NOTE: The number of macro-instruction identification numbers that are printed is limited to 28. The list includes the macro ID's of the first 27 control macro-instructions encountered and of the last control macro-instruction encountered. Macro ID's are not printed for any macro-instructions encountered after the twenty-seventh macro-instruction and before the last macro-instruction in the list.

Error Message Lines

Error message lines diagnose errors detected during program testing and output editing. In the case of execution errors, the first line identifies the macro-instruction that caused the error, or that detected an error in the problem program. This line includes the associated output selection code, if any, but the error message is printed regardless of the output selection codes used in selecting data for editing.

The printing format for error messages is as follows:

```
P) MACRO ID NNN, ERROR
CCCCCCC EEEEMMMM
```

P)
is the output selection code (if any) associated with the macro-instruction.

NNN
is the macro-instruction identification number assigned by the assembler to the macro-instruction that caused or detected the error.

CCCCCCC
is a standard operating system error message code.

EEEEMMMM
is the error message.

SAMPLE TEST PROGRAM AND TEST OUTPUT

This section describes the use of a series of TESTRAN macro-instructions to test the operation of a problem program. The purpose of this program is to read a series of punched cards, to alter symbols contained in certain card images, and to punch the modified images. The processing portion of this program, a control section named SYMALTER, is shown in Figure 7 together with a related series of TESTRAN macro-instructions.

```
BEGIN    TEST  OPEN,START,JOB1,SELECT=1
         TEST  DEFINE,FLAG,CARD4
         TEST  AT,READ+14                  CHECK ON INPUT
         TEST  ON,0,4,4,A                  IS THIS THE 4TH CARD
         GO    BACK                        NO, GO BACK TO PROBLEM
A        DUMP  DATA,INAREA,SELECT=2        YES, DUMP INPUT CARD IMAGE
         DUMP  PANEL,(G'4',G'8'),SELECT=4    AND PANEL
         SET   FLAG,CARD4,=1               SET CARD 4 INDICATOR TO 1
         GO    TO,SEEOUT                   PICK UP FLAGS
         TEST  AT,RETURN1                  CHECK ON OUTPUT
         TEST  WHEN,CARD4,B                IS THIS THE 4TH CARD
         GO    BACK                        NO, GO BACK TO PROBLEM
B        DUMP  DATA,OUTAREA,SELECT=3       YES, DUMP OUTPUT CARD IMAGE
         SET   FLAG,CARD4,=0               SET CARD 4 INDICATOR TO 0
SEEOUT   DUMP  DATA,ERRFLAG,OUTAREA        DUMP FLAGS
         GO    BACK                        CONTINUE
*
*
SYMALTER CSECT
START    BALR  10,0                     ***
         USING *,10                       * INITIALIZE
         LA    8,OUTAREA                  * ROUTINE
         LA    6,TABLE                    *
         B     READ                     ***
*
PUNCH    LA    8,OUTAREA                ***
         LA    9,=V(PCHCRD)               *
         BALR  15,9                       * PUNCH CARD
RETURN1  MVC   OUTAREA(88),STARTO       ***
*
READ     LA    4,INAREA                 ***
         LA    9,=V(REDCRD)               *
         BALR  15,9                       * GET NEW CARD
         BC    15,EOD                     * FROM INPUT
         SR    7,7                        *
         CLI   0(4),X'FO'                 *
         BC    4,CHKEOC1                  *
         LA    3,TABLE                    * BUILD SYMBOL
CMPTAB   CLC   9(8,4),10(3)               * SUBSTITUTION
         BC    8,REENTER                  * TABLE IF A
         LA    3,18(3)                    * CONTROL CARD
         CLR   3,6                        * IS DETECTED
         BC    4,CMPTAB                   *
         STH   7,0(6)                     *
         MVC   1(17,6),0(4)               *
         NI    1(6),X'OF'                 *
         LA    6,18(6)                    *
         ST    6,ENDTAB                   *
         BC    15,READ                  ***
*
REENTER  STH   7,0(3)                   ***
         MVC   1(17,3),0(4)               * ALTER SYMBOL
         NI    1(3),X'OF'                 * IN SUB TABLE
         BC    15,READ                  ***
*
CHKEOC1  BCTR  4,0                      ***
         BCTR  8,0                        * BUILD OUTPUT
CHKEOC   LA    5,0(8,7)                   * CARD ONE
         LA    3,0(4,7)                   * CHARACTER
         CL    5,ENDOUT                   * AT A TIME
         BC    8,PUNCH                    *
```

Figure 7. Sample Test Program

(continued)

```
         MVC   0(1,5),0(3)              * CHECK FOR
         CLI   1(3),X'CO'               * END OF CARD
         BC    2,BUMPCNT                *
         CLI   0(3),X'CO'               *
         BC    4,BUMPOUT               ***
*
         LA    3,TABLE                 ***
COMP     CLC   10(8,3),1(8)             * CHECK LAST
         BC    8,NEWSYM                 * COMPLETE
         LA    3,18(3)                  * SYMBOL
         CL    3,ENDTAB                 * AGAINST TABLE
         BC    4,COMP                  ***
*
         LA    8,1(8,7)                ***
RESET    LA    4,1(4,7)                 * SYMBOL OK,
ZEROCNT  SR    7,7                      * NO CHANGE
         BC    15,CHKEOC               ***
*
*                                      ***
BUMPCNT  LA    7,1(7)                   * GET SYMBOL
         BC    15,CHKEOC                * LENGTH
*                                      ***
*
BUMPOUT  LA    8,1(8)                  ***
         LA    4,1(4)                   * GO TO SYMBOL
         BC    15,CHKEOC               ***
*
NEWSYM   MVC   1(8,8),2(3)             ***
         AH    8,0(3)                   * SUBSTITUTE
         LA    8,1(8)                   * NEW SYMBOL
         B     ZEROCNT                 ***
*
*
*                                      ***
EOD      B     OUT                      * FINAL EXIT
*                                      ***
*
*
ENDOUT   DC    A(OUTAREA+80)
ERRFLAG  DC    C'*'
*
*
STARTIN  DC    C' '
STARTO   DC    C' '
OUTAREA  DC    CL88' '
INAREA   DC    CL88' '
TABLE    DC    180H'0'
ENDTAB   DC    A(*)
END      EQU   *
*
         END   START
*                                       ** LINKAGE EDITOR INPUT
*        ENTRY BEGIN                     * INCLUDES THIS STATEMENT
*                                       ** WHEN PROGRAM IS TESTED
```

Figure 7. Sample Test Program

The macro-instructions shown in Figure 7 cause the TESTRAN interpreter to perform the following actions when the fourth card image is processed:

1. When the input subroutine REDCRD returns control to READ+14 in SYMALTER, the TESTRAN interpreter records the contents of:

   - The input buffer INAREA.
   - The general registers 4 and 8.
   - The data fields at ERRFLAG, STARTIN, and STARTO.

2. When the output subroutine PCHCRD returns control to RETURN1 in SYMALTER, the TESTRAN interpreter records the contents of:

   - The output buffer OUTAREA.
   - The data fields at ERRFLAG, STARTIN, and STARTO.

Printed test output related to these actions is shown in Figure 8. The individual macro-instructions are discussed in the succeeding paragraphs.

Macro ID 000 is assigned to the following macro-instruction:

```
BEGIN  TEST    OPEN,START,JOB1,SELECT=1
```

Problem program execution begins when the control program passes control to BEGIN, as specified in a linkage editor ENTRY control statement. This macro-instruction causes the symbol JOB1 to be printed in the test output page heading; it also generates the output line that appears beneath the heading of the first page shown in Figure 8. The LINK option is assumed, and values specified by the installation at system generation time are assumed for the omitted MAXP and MAXE operands. An output selection code of 1 is implied for all test data for which no output selection code is specified by other macro-instructions. BEGIN inserts TESTRAN SVC's at READ+14 and RETURN1, as specified by the two TEST AT macro-instructions, and passes control to the location START in SYMALTER.

Macro ID 001 is assigned to the following macro-instruction:

```
       TEST    DEFINE,FLAG,CARD4
```

This macro-instruction causes a logical flag named CARD4 to be assembled with an initial condition of 0. It produces no printed output.

Macro ID's 002, 003, and 004 are assigned to the following macro-instructions:

```
       TEST    AT,READ+14
       TEST    ON,0,4,4,A
       GO      BACK
```

When the input subroutine REDCRD returns control to READ+14 in SYMALTER, execution of the inserted TESTRAN SVC causes an unnamed counter to be incremented and tested by the TEST ON macro-instruction. On the fourth return from REDCRD, an affirmative test result causes A to be the next macro-instruction encountered. On all other returns from REDCRD, a nonaffirmative test result causes the GO BACK macro-instruction to execute the displaced problem program instruction (SR7,7) and to pass control to the instruction at READ+16; no printed output results, because no action macro-instruction is encountered.

Macro ID's 005 and 006 are assigned to the following macro-instructions:

```
A      DUMP    DATA,INAREA,SELECT=2
       DUMP    PANEL,(G'4',G'8'),SELECT=4
```

When encountered following the fourth return from REDCRD, these macro-instructions record the contents of the input buffer INAREA and the contents of general registers 4 and 8. Editing of this data causes printing of output lines for TEST AT (macro ID 002) and TEST ON (macro ID 003), as shown in Figure 8. Following these lines is the recorded buffer and register data, identified by macro ID's 005 and 006.

```
  JOB1                          TESTRAN OUTPUT               DATE 10/164                TIME 10/04          PAGE    1

1) MACRO ID 000, TEST OPEN   , TESTRAN CONTROL SECTION = BEGIN   , IDENTIFICATION JOB1


AT LOCATION         (SYMALTER) 0000EC 0100EC ENTER BEGIN


EXECUTED STATEMENTS, BEGIN     003


2) MACRO ID 005, DUMP DATA    STARTING IN SECTION SYMALTER
  0154     INAREA
010154     COMEBACK MVC   WRITAREA(88),ENTER       CLEAR BUFFER FOR NEXT CARD          0003


4) MACRO ID 006, DUMP PANEL
G'04' 00010154 G'08' 000100FC
PSW  00 0 1 0002 0 0 01008C  CC=0  FIX POINT OVERFLOW OFF  DEC OVERFLOW OFF  EXP UNDERFLOW OFF  SIGNIFICANCE OFF


EXECUTED STATEMENTS, BEGIN     007, 008


1) MACRO ID 014, DUMP DATA    STARTING IN SECTION SYMALTER
  00F8     ERRFLAG     STARTIN     STARTO
0100F8      *


AT LOCATION RETURN1 (SYMALTER) 0000DA 0100DA ENTER BEGIN


EXECUTED STATEMENTS, BEGIN     010


3) MACRO ID 012, DUMP DATA    STARTING IN SECTION SYMALTER
  00FC     OUTAREA
0100FC     COMEBACK MVC   WRITAREA(88),ENTER       CLEAR BUFFER FOR NEXT CARD          0003


EXECUTED STATEMENTS, BEGIN     013


1) MACRO ID 014, DUMP DATA    STARTING IN SECTION SYMALTER
  00F8     ERRFLAG     STARTIN     STARTO
0100F8      *           1


***  IEGE07  END OF TESTRAN EDIT--0000005 STATEMENTS PROCESSED
```

Figure 8. Sample Test Output

Macro ID's 007 and 008 are assigned to the following macro-instructions:

```
        SET     FLAG,CARD4,=1
        GO      TO,SEEOUT
```

The flag CARD4 is set to the logical condition 1, indicating that the fourth card has been read. The next macro-instruction encountered is SEEOUT, which records the contents of the data fields at ERRFLAG, STARTIN, and STARTO before control is returned to the problem program. Editing of the data recorded by SEEOUT causes printing of an output line for SET FLAG (macro ID 007) and GO TO (macro ID 008) as shown in Figure 8.

Macro ID's 009, 010, and 011 are assigned to the following macro-instructions:

```
        TEST    AT,RETURN1
        TEST    WHEN,CARD4,B
        GO      BACK
```

When the output subroutine PCHCRD returns control to RETURN1 in SYMALTER, execution of the inserted TESTRAN SVC causes the logical flag CARD4 to be tested by the TEST WHEN macro-instruction. On the fourth return from PCHCRD, an affirmative test result causes B to be the next macro-instruction encountered, and CARD4 is reset to logical condition 0 before control is returned to the problem program. On all other returns from PCHCRD, a nonaffirmative test result causes the GO BACK macro-instruction to execute the displaced problem program instruction (MVC OUTAREA(88),STARTO) and to pass control to READ; no printed output results, because no action macro-instruction is encountered.

Macro ID 012 is assigned to the following macro-instruction:

```
B       DUMP    DATA,OUTAREA,SELECT=3
```

When encountered following the fourth return from PCHCRD, this macro-instruction records the contents of the output buffer OUTAREA. Editing of this data causes printing of output lines for TEST AT (macro ID 009) and TEST WHEN (macro ID 010) as shown in Figure 8. Following these lines is the recorded buffer data, identified by macro ID 012.

Macro ID 013 is assigned to the following macro-instruction:

```
        SET     FLAG,CARD4,=0
```

The flag CARD4 is reset to the logical condition 0, indicating that the modified image of the fourth card has been punched. The next macro-instruction encountered is SEEOUT; editing of the data recorded by this action macro-instruction causes printing of an output line for SET FLAG (macro ID 013) as shown in Figure 8.

Macro ID's 014 and 015 are assigned to the following macro-instructions:

```
SEEOUT DUMP     DATA,ERRFLAG,OUTAREA
       GO       BACK
```

SEEOUT records the contents of the main storage area from ERRFLAG through OUTAREA-1; this area includes the data fields ERRFLAG, STARTIN, and STARTO. The recorded data is shown in Figure 8, identified by macro ID 014. The GO BACK macro-instruction executes the instruction displaced by the last executed TESTRAN SVC and returns control to the next sequential problem program instruction.

JOB ORGANIZATION

Full use of TESTRAN requires the execution of four job steps: assembly, linkage editing, program testing, and output editing. Each job step can be performed as a separate job or can be combined with other job steps in a single job; in either case, the job steps must be performed in logical order. For each job, specific job control statements are required. The programmer can write either a separate EXEC control statement (and associated DD control statements) for each job step to be performed or, alternatively, a single EXEC control statement for a previously cataloged procedure that includes all necessary control statements for one or more job steps. The basic control statements required for each job step are shown in Table 36. For detailed descriptions of job control statements, refer to the publication IBM System/360 Operating System: Job Control Language; for a description of cataloged procedures, refer to the publication IBM System/360 Operating System: System Programmer's Guide, Form C28-6550.

Table 36. Job Control Statements Required for Assembly, Linkage Editing, Program Testing, and Output Editing

Name	Operation	Operand	Comments
$stepname_1$	EXEC	operand	assembler job step
ddname	DD	operand	input data set (source module)
SYSLIN	DD	operand	output data set (object module)
	etc.		
$stepname_2$	EXEC	operand	linkage editor job step
SYSLIN	DD	operand	primary input data set (object module)
SYSLIB	DD	operand	call library
SYSUT1	DD	operand	buffer data set
SYSLMOD	DD	operand	output data set (load module)
SYSPRINT	DD	operand	log output
ddname	DD	operand	additional library
ddname	DD	operand	additional library
	etc.		
$stepname_3$	EXEC	operand	problem program job step
SYSTEST	DD	operand	output data set (unedited test data)
ddname	DD	operand	problem program data set
ddname	DD	operand	problem program data set
	etc.		
stepname	EXEC	operand	TESTRAN editor job step
SYSTEST	DD	operand	input data set (unedited test data)
SYSUT1	DD	operand	intermediate data set
SYSPRINT	DD	operand	output data set (edited test data)

Execution of the assembler processing program produces an object module, which will include a source program symbol table if the appropriate option is specified. The linkage editor produces a load module from one or more object modules; if specified in the EXEC control statement, the symbol table is included in the load module and passed to the TESTRAN editor. Dynamic testing occurs as a result of execution of the problem program, providing that control is passed to a TEST OPEN macro-instruction. Execution of the TESTRAN editor causes data to be printed according to options specified in the EXEC control statement and the formats defined in the symbol table.

ASSEMBLER OPTION: Unless data modifiers are always included in the TESTRAN macro-instructions, the assembler symbol table option should

always be specified in the PARM field of the EXEC control statement for the assembler. This option is specified as

PARM='TEST'

To ensure proper editing of test data, the symbol table option should be specified for each object module that is included in a load module to be tested, or that is included in any other load module to be executed under the same task.

LINKAGE EDITOR OPTION: The test option must be specified in the PARM field of the EXEC control statement for the linkage editor, to cause the symbol table (if present) and the original external symbol dictionaries produced by the assembler to be included in the load module. This option is specified as

PARM='TEST'

To ensure proper editing of test data, the test option should be specified for each load module to be tested, and for all other load modules that are to be executed under the same task.

TESTRAN EDITOR OPTIONS: Two options can be specified in the PARM field of the EXEC statement for the TESTRAN editor. This field has the general form

PARM='TaPb'

where a is either an unsigned decimal integer from 1 to 8, a blank, or the letter A, and where b is an unsigned three-digit decimal integer. Their use is as follows:

- The value a specifies the output selection code or codes associated with test data to be edited; the field Ta can be repeated as many times as required to select desired test data during a single execution of the TESTRAN editor. The value a, if an integer, specifies a single output selection code; if a blank, it specifies that data with no associated output selection code is to be edited. All test data is edited if a is the letter A or if the field Ta is omitted.

- The integer b specifies the maximum number of pages to be printed. This number must not be greater than the maximum page count established at the installation at system generation time. Any maximum count specified in the TEST OPEN macro-instruction is overridden. If no maximum count is specified in either the EXEC control statement or the TEST OPEN macro-instruction, the installation maximum count is assumed.

DATA DEFINITION: The TESTRAN interpreter requires a data set for storage of intermediate output; this data set later becomes input to the TESTRAN editor. The necessary data control blocks are generated and opened automatically, but they must be completed by DD control statements placed after the EXEC control statements for both the problem program and the TESTRAN editor. These DD control statements must specify that the data set is to be retained until all desired test data has been edited.

The TESTRAN editor requires two additional data sets; one for working storage and one for edited output. The data control blocks that the TESTRAN editor creates for these data sets must be completed by DD control statements with the names SYSUT1 and SYSPRINT, respectively. These DD control statements must follow the EXEC control statement for the TESTRAN editor. Neither data set should be retained after completion of the job step.

Positional operands and the optional values of keyword operands can be written in a variety of forms, as specified by value mnemonics. Some of these forms are conventional assembler language expressions and addresses; however, certain restrictions, stated in this appendix, must be followed when these forms are used in system macro-instructions. Other forms are new. Their use in system macro-instructions is defined in detail.

This appendix also describes the assembler language instructions that become part of the macro-expansion when the appropriate operand form is used in a supervisor or data management macro-instruction.

DESCRIPTIONS OF OPERAND FORMS

The allowable operand forms, and the value mnemonics that specify them, are shown in Table 37. The forms other than symbol, integer, and coded value are described below.

Table 37. Operand Forms and Related Value Mnemonics

Operand Form	Value Mnemonic that Specifies the Operand Form
Symbol	symbol
Relocatable expression	relexp addr
Implied address	addrx addx adval
Explicit address	addrx addx adval
Integer (self-defining decimal value)	integer
Absolute expression	absexp value
Register notation	addr addrx value
TESTRAN register notation	adval
Character constant	text
Coded value	code
Data attribute notation	tls

RELOCATABLE EXPRESSION

A relocatable expression can be a single relocatable term (symbol), or it can be an arithmetic combination of relocatable and absolute terms, such as symbols and self-defining values. The expression must reduce to a single relocatable value, which can be a complex relocatable value.

The expression is evaluated as an A-type address constant, and therefore cannot contain a literal.

Examples of relocatable expressions are as follows:

```
S             S is relocatable.
OUT-3*PT      OUT is relocatable; PT is absolute.
```

NOTES: The following notes apply:

- Individual symbolic terms and the evaluated address need not be addressable by means of a base register at execution time. There are no special restrictions on the use of terms that also appear as operands of an EXTRN statement.

- Relocatable expressions written as operands of supervisor and data management macro-instructions must not begin with a left parenthesis and end with a right parenthesis. This restriction results from the form of register notation used in these macro-instructions, and therefore does not apply to TESTRAN macro-instructions.

- In supervisor and data management macro-instructions, a reference to the location counter refers to the location of the assembled parameter (A-type address constant). A reference to the location counter must not be made in a relocatable expression written as an operand of a TESTRAN macro-instruction.

IMPLIED ADDRESS

An implied address can be any indexed or non-indexed implied address allowed by the assembler language.

Examples of implied addresses are as follows:

```
S(X)          S is absolute or relocatable;
              X is absolute.
BUF-3(X1)     BUF is absolute or relocatable;
              X1 is absolute.
1000
=F'5'
```

NOTES: The following notes apply to implied addresses written in supervisor and data management macro-instructions:

- The address must be addressable by means of a base register specified in a USING statement and loaded by the problem program. This restriction applies except when the address is absolute and has a value less than 4096.

- The address must be either absolute or simply relocatable. That is, no more than one relocatable term can be unpaired. Paired relocatable terms have opposite signs and are symbols defined in a single

control section; the expression formed by paired relocatable terms is therefore absolute.

- A symbol that also appears as an operand in an EXTRN statement can be written, providing that it is the only relocatable term in the expression. The external symbol RTN, for example, must be the operand of a USING statement provided by the programmer, and the address of RTN must be in the register specified by the USING statement.

- The operand must not begin with a left parenthesis and end with a right parenthesis. This restriction results from the form of register notation used in these macro-instructions.

- The effective address is formed by a LA instruction in the macro-expansion. A reference to the location counter refers to the location of this LA instruction.

The following notes apply to implied addresses written in TESTRAN macro-instructions:

- The address expression of the operand is assembled as an A-type address constant. Therefore, the address need not be addressable.

- A reference to the location counter must not be made.

- An operand written as a literal results in a data constant that is part of the macro-expansion.

EXPLICIT ADDRESS

An explicit address can be indexed or non-indexed, but must be written as in an RX-format assembler language instruction. That is, two absolute expressions of the form (X,B) must be written, thereby designating both an index register and a base register. If the address is non-indexed, X must not be omitted but must be written as either a zero or an equivalent absolute expression.

Examples of explicit addresses are as follows:

D(X,B) 12(0,3)	D,X, and B are absolute.
TAB-YY(R+2,N2)	TAB and YY are both either absolute or relocatable and defined in the same control section; R and N2 are absolute.

NOTES: The following notes apply:

- Explicit addresses written as operands of supervisor and data management macro-instructions must not begin with a left parenthesis and end with a right parenthesis. A displacement must always be written, even if its value is zero. This restriction results from the form of register notation used in supervisor and data management macro-instructions, and therefore does not apply to TESTRAN macro-instructions.

- In supervisor and data management macro-instructions, the explicit address is used as the operand of a L or LA instruction in the macro-expansion. A reference to the location counter refers to the location of this instruction. A reference to the location counter must not be made in an explicit address written as an operand of a TESTRAN macro-instruction.

ABSOLUTE EXPRESSION

An absolute expression can be a single absolute term, or it can be an arithmetic combination of relocatable and absolute terms, such as symbols and self-defining values. The expression must reduce to a single absolute value.

The expression is evaluated as an A-type address constant, and therefore cannot contain a literal.

Examples of absolute expressions are as follows:

```
5
LOB-S          LOB and S are both either absolute or relocatable
               and defined in one control section.
X'FF'+2
```

NOTES: The following notes apply:

- Absolute expressions written as operands of supervisor and data management macro-instructions must not begin with a left parenthesis and end with a right parenthesis. This restriction results from the form of register notation used in these macro-instructions, and therefore does not apply to TESTRAN macro-instructions.
- In supervisor and data management macro-instructions, a reference to the location counter refers to the address of the assembled parameter (A-type address constant). References to the location counter cannot be made in absolute expressions written as operands of TESTRAN macro-instructions.

REGISTER NOTATION

Register notation is a way of designating a register that contains, at execution time, the address or value parameter specified by the operand. This register must be loaded with the parameter by the problem program. The macro-expansion stores the contents of the designated register in the parameter list, or loads the contents into a parameter register.

An operand that begins with a left parenthesis and ends with a right parenthesis, and that according to the format description must not be a sublist, is assumed to be an absolute expression designating a register that contains the parameter. The parentheses need not be paired.

Examples of register notation are as follows:

```
(R)            R is absolute.
(5)
(LOB-S)        LOB and S are both either absolute or relocatable
               and defined in one control section.
```

The parameter must be contained in the rightmost bits of the register, and the leftmost unused bits of the register must all be zeros, except when an individual macro-instruction description states otherwise.

In an addrx operand, the effect of using register notation can be achieved by writing an implied address of the form 0(B), or an explicit address of the form 0(0,B), where register B contains an existing

effective address. This should not be done, because the macro-expansion will contain:

- A L or LA instruction instead of a LR instruction, in the case of an R-type macro-instruction.
- An unneccessary LA instruction, in the case of the E form of an S-type macro-instruction.

In either of the above two cases, the macro-expansion will require more space and execution time.

NOTE: If the expression (A+B)*(A-B), or any like it, is not intended to be assumed to be register notation, it should be prefixed by 0+.

TESTRAN REGISTER NOTATION

TESTRAN register notation is a way of designating either a register that contains a value at execution time, or a register or series of registers whose contents are to be recorded.

A single register is specified by the letter G (for general register) or the letter F (for floating-point register) followed by a symbol or decimal integer enclosed in single quotation marks. A symbol, if used, must be a symbolic name equated to a register number. An integer must be unsigned and a valid number for a register of the type specified; it can include a single high-order zero if the number of the register is less than 10.

A series of registers is specified in the same way, except that the numbers or symbolic names of two registers, separated by a comma, are specified within a single pair of single quotation marks. The two explicitly specified registers are the first and last registers in the specified series.

TESTRAN register notation is used in certain operands that can also be written in the form of an indexed implied address. The special form of the notation indicates that the register contains a value rather than an address.

Examples of TESTRAN register notation are as follows:

G'2'	
G'SUM'	SUM is a symbolic name equated to the number of a general register.
F'2'	
F'PRODUCT'	PRODUCT is a symbolic name equated to the number of a floating-point register.
G'0,4'	The series includes general registers 0, 1, 2, 3, and 4.
G'14,2'	The series includes general registers 14, 15, 0, 1, and 2.
F'0,VAL'	VAL is a symbolic name equated to the number of a floating-point register. This series includes floating-point registers 0, VAL, and any floating-point registers lower numbered than VAL.
F'6,2'	The series includes floating-point registers 6, 0, and 2.

CHARACTER CONSTANT

A character constant specifies a message or comment to be written on a console or printer.

A character constant must be written with enclosing single quotation marks; this is shown explicitly when the text value mnemonic appears in a format description. The entire operand is identical to the constant subfield of an assembler language DC data definition instruction. The specified message or comment does not include the enclosing quotation marks.

Two single quotation marks or ampersands, unseparated by other characters, must be included in the character constant for each single quotation mark or ampersand to be included in the message or comment.

Examples of character-constant operands are as follows:

'MOUNT XT04 && YA77'	The message is printed as: MOUNT XT04 & YA77
'''NETPAY'' IS NEGATIVE'	The comment is printed as: 'NETPAY' IS NEGATIVE

NOTE: The maximum length of a message specified by a supervisor macro-instruction cannot exceed the line length of the console output device having the shortest line length. The maximum length of a comment specified by a TESTRAN macro-instruction is always 120 characters.

DATA ATTRIBUTE NOTATION

Data attribute notation is a way of designating type, length, and scale attributes to be used in comparing, modifying, or editing data. These attributes override any attributes defined in the symbol table for the same data.

Data attribute notation is similar in format to the type and modifier subfields of operands of DC and DS data definition instructions. The notation is written as:

tLxSy

where t is a type code, Lx is a length modifier, and Sy is a scale modifier. The type code and modifiers are each optional, but if present must be written in the order shown. Modifiers are written as the mnemonic character L or S followed by an appropriate value for x or y; x is written as an unsigned decimal integer, y as a signed or unsigned decimal integer. Allowable type codes and corresponding substitutable values for x and y are listed in Table 38.

The length and scale attributes, unless explicitly specified, are implied by the type code as shown in Table 38. If the type code is omitted, the type is assumed to be hexadecimal, and the length, unless explicitly specified, is assumed to be 1 byte (when the data is to be recorded, compared, or modified) or 4 bytes (when the data is to be edited).

Examples of data attribute notation are as follows:

XL128	
FS-2	The scale factor is -2.
EL8S14	The maximum scale factor for the data type and length is shown.
I	The implied length varies with the format of the instruction.

Table 38. Data Attribute Specifications

Type Code (t)	Length in Bytes[1]		Scale Factor[2] (y)
	Explicit (x)	Implied	
C (character)	1-255	1	(not applicable)
X (hexadecimal)	1-255	1	(not applicable)
B (binary)	1-255	1	(not applicable)
H (fixed point)	1-8	2	-187 to +346
F (fixed point)	1-8	4	-187 to +346
E (floating point)	1-8	4	(not applicable)
D (floating point)	1-8	8	(not applicable)
P (packed decimal)	1-16	1	(not applicable)
Z (zoned decimal)	1-16	1	(not applicable)
I (instruction)	(not applicable)	(variable)	(not applicable)

[1] The implied length is used if the type code (t) is present but the length modifier (Lx) is omitted.
[2] The implied scale attribute is zero if the scale modifier (Sy) is omitted.

OPERAND PROCESSING

The processing of operand forms in supervisor and data management instructions is discussed below to show the significance of value mnemonics, and to aid in the understanding of macro-expansions.

In this discussion, a reference to an S-type macro-instruction refers to only the standard form of the macro-instruction. References to L and E forms of the macro-instruction are made explicitly (as L-form or E-form).

Note that an addr operand of an S-type macro-instruction can be written in only the relexp form in an L-form macro-instruction, and it can be written in the addrx forms in an E-form macro-instruction. Also, a value operand can be written in only the absexp form in an L-form macro-instruction.

SYMBOL: The symbol is assembled as an eight-byte character constant.

- In an S-type or L-form macro-instruction, the character constant is part of the resulting parameter list.
- In an E-form macro-instruction, the character constant results from literal notation used in a MVC instruction. This instruction moves the character constant to the remote parameter list.

RELEXP: In an L-form macro-instruction, the relocatable expression is assembled as an A-type address constant that is part of the resulting parameter list.

ADDR - RELOCATABLE EXPRESSION: In an S-type or L-form macro-instruction, the relocatable expression is assembled as an A-type address constant that is part of the resulting parameter list.

ADDR - REGISTER NOTATION: In an S-type macro-instruction, register notation results in a ST instruction that stores the existing effective address into the resulting parameter list.

ADDRX - IMPLIED OR EXPLICIT ADDRESS: An implied or explicit address results in a L or LA instruction that loads a register with an effective address.

- In an R-type macro-instruction, the L or LA instruction loads a parameter register.
- In an E-form macro-instruction, the L or LA instruction loads a working register (14, 15, or 0), and a ST instruction (which is also part of the macro-expansion) stores the address into the remote parameter list.

ADDRX - REGISTER NOTATION: Register notation results in the following:

- In an R-type macro-instruction, a L or LR instruction that loads a parameter register (unless special register notation is used).
- In an E-form macro-instruction, a ST instruction that stores the existing effective address into the remote parameter list.

ABSEXP: In an L-form macro-instruction, the absolute expression is assembled as an A-type address constant that is part of the resulting parameter list.

VALUE - ABSOLUTE EXPRESSION: In an S-type or L-form macro-instruction, the absolute expression is assembled as an A-type address constant that is part of the resulting parameter list.

In an R-type or E-form macro-instruction, if the absolute expression can have a value whose absolute magnitude is equal to or greater than 4096, the expression is assembled as an A-type address constant. This constant results from literal notation used in a L instruction.

- In an R-type macro-instruction, the L instruction loads a parameter register.
- In an E-form macro-instruction, the L instruction loads a working register (14, 15, or 0), and a ST instruction (which is also part of the macro-expansion) stores the value into the remote parameter list.

If the absolute magnitude must be less than 4096, a LA instruction results, instead of a L instruction. The instruction operand is of the form D(0,0), where D is the absolute expression. Processing then occurs as for a L instruction.

VALUE - REGISTER NOTATION: Register notation results in the following:

- In an R-type macro-instruction, a LR instruction results that loads the value into a parameter register (unless special register notation is used).
- In an S-type macro-instruction, a ST instruction results that stores the value into the resulting parameter list.
- In an E-form macro-instruction, a ST instruction results that stores the value into the remote parameter list.

APPENDIX B: L AND E FORMS OF S-TYPE MACRO-INSTRUCTIONS

L- AND E-FORM MACRO-EXPANSIONS

The L form of an S-type macro-instruction results in a macro-expansion consisting of only a parameter list.

The E form of an S-type macro-instruction results in a macro-expansion consisting of only executable instructions. These instructions perform the following functions:

- Load the parameter list register with the address of a remote parameter list; alternatively, the user's problem program may have loaded the address of a remote parameter list into the parameter list register before execution of the macro-instruction.
- Store parameters in the remote parameter list, if any parameters need be modified at execution time. These parameters can be loaded into registers by the user's problem program before execution of the macro-instruction, or they can be dynamically formed by the macro-expansion (by a load address (LA) instruction, for example).
- Give control to the required control program routine. This is done by execution of an SVC or branch instruction, just as in the standard form of the macro-instruction. The control program routine refers to the remote parameter list pointed to by the parameter list register.

USE OF L- AND E-FORM MACRO-INSTRUCTIONS

The L and E forms of an S-type macro-instruction can be used together to request a control program service that would otherwise be requested by the standard form of the macro-instruction. This facility is advantageous for the following reasons:

- Use of an E-form macro-instruction allows use of the addrx operand forms instead of the addr operand forms.
- If a particular macro-instruction is to be written more than once in a program, use of two or more E-form macro-instructions and one L-form macro-instruction conserves main storage. All of the E-form macro-instructions should refer to the parameter list that results from the L-form macro-instruction. If they do, two or more physically distinct parameter lists are not required, and main storage is conserved. Any of the parameters in the list can be modified when each E-form macro-instruction is executed.
- If the parameter list of a macro-instruction must be modified at execution time, use of an E-form macro-instruction allows the macro-instruction to be written in a reenterable program. The E-form macro-instruction must refer to a remote parameter list that is in either a higher level program or dynamically allocated storage.

Proper use of these macro-instruction forms requires understanding of the following:

- How the L and E forms of an S-type macro-instruction are specified.
- The different operand forms allowed in these macro-instruction forms.
- The action of a combination of one L-form and one E-form macro-instruction.
- Special requirements of operands written in L- and E-form macro-instructions.

THE MF KEYWORD OPERAND

The L and E forms of an S-type macro-instruction are specified by an extra keyword operand in the macro-instruction. The keyword of this operand is MF (for macro-form).

The L form is specified by:

```
MF=L
```

The optional value is L. A symbol written in the name field of the macro-instruction can be used to refer to the resulting parameter list.

The E form is specified by an operand having the following representation:

$$MF=(E,\left\{\begin{matrix}\text{pl-addrx}\\(1)\end{matrix}\right\})$$

The optional value is a sublist consisting of E and the address of a remote parameter list. This address is loaded, by the macro-expansion, into register 1 (the parameter list register). By writing the special register notation (1), the programmer specifies that the address of the parameter list will have been loaded into register 1 before execution of the macro-instruction.

OPERAND FORMS USED IN L- AND E-FORM MACRO-INSTRUCTIONS

Address and value operands can be written in a standard S-type macro-instruction in the forms specified by the value mnemonics, addr and value. Different operand forms are allowed in the L and E forms of the macro-instruction, as follows:

- In an L-form macro-instruction, an address operand can be written in only the relexp form. A value operand can be written in only the absexp form.
- In an E-form macro-instruction, an address operand can be written in the addrx form. A value operand can still be written in the value form.

Note that in an R-type macro-instruction, address operands can be written in the addrx forms. Not allowing addrx forms in the standard form of an S-type macro-instruction reduces the time required by the assembler to process the macro-instruction.

Refer to Appendix A for more information about why different operand forms are allowed.

OPERAND COMBINATIONS

The format description of an S-type macro-instruction shows how operands in the standard form of the macro-instruction are to be written. Its interpretation is different when the L or E form of the macro-instruction is being written. In these cases, the format description shows the combination of operands that is to be written; a particular operand can ordinarily be written in either the L-form macro-instruction or the E-form macro-instruction.

Examples of L- and E-form macro-instruction use are presented below. These examples use the WTOR (write to operator with reply) macro-instruction, which has the following format description:

Name	Operation	Operand
[symbol]	WTOR	message-'text',reply-addr,length-value ,ecb-addr

The message operand specifies the message to be written on the operator's console.

The reply operand specifies the address of an area in the program in which the reply message will be placed. The length operand specifies the length of this area.

The ecb operand specifies the address of an event control block (ECB). The program should issue a WAIT macro-instruction referring to this event control block to determine when the message reply has been received.

In the following examples, EX1 is the standard form of the S-type macro-instruction. It results in the message FINISHED being written on the operator's console. The reply is to be placed at location LOC1, and is to have a length of 15 bytes. The program should refer to event control block ECB1 to determine when the message reply has been received.

EX2A is an L-form macro-instruction in which the reply operand is omitted. EX2B is an E-form macro-instruction referring to the parameter list resulting from the EX2A macro-instruction. The reply operand is present in EX2B, and, when that macro-instruction is executed, the effect of the macro-instruction is exactly the same as that of EX1. (The reply operand in EX2A and the message operand in EX2B are delimited by commas, because a positional operand is written to the right of each.)

EX3 also refers to the EX2A parameter list. The example shows that the reply operand in EX3 can be written as an indexed implied address (allowed by the addrx form), and that the length parameter in the EX2A parameter list can be changed at execution time.

EX4A is an L-form macro-instruction in which all necessary operands are written. EX4B is an E-form macro-instruction in which only the MF operand is written. When EX4B is executed, the address of a parameter

list (for example, EX4A) must have been loaded into register 8 by the user's problem program.

```
EX1     WTOR    'FINISHED',LOC1,15,ECB1
EX2A    WTOR    'FINISHED',,15,ECB1,MF=L
EX2B    WTOR    ,LOC1,MF=(E,EX2A)
EX3     WTOR    ,LOC1(5),25,MF=(E,EX2A)
EX4A    WTOR    'LAST ONE',LOC4,50,ECB4,MF=L
EX4B    WTOR    MF=(E,(8))
```

NOTES: The E forms of most S-type macro-instructions (but not of the WTOR macro-instruction) can contain all operands. Writing all operands in an E-form macro-instruction eliminates the need for an L-form macro-instruction; the remote parameter list can be an unused assembled area of the user's problem program, or it can be an area of allocated main storage. The operands written in the E-form macro-instruction result in parameters that are stored in the remote parameter list. However, operands should be written in E-form macro-instructions only when the corresponding parameters must be formed or changed at execution time; otherwise, unnecessary instructions will be generated and executed.

If a parameter list must be formed by a reenterable program, and if some but not all of the parameters in the list are to be modified before each of two or more executions of an E-form macro-instruction that refers to the list, the following technique can be used. The program can contain an L-form macro-instruction that results in a parameter list containing the constant parameters. When it is executed, the program can move this parameter list to allocated main storage. The E-form macro-instruction can then refer to the remote list without execution of instructions that unnecessarily modify the constant parameters.

ORDINARY AND SPECIAL OPERAND REQUIREMENTS

The ordinary operand requirements of L- and E-form macro-instructions, stated in the preceding topics, are reviewed below briefly:

- The MF keyword operand must be written to specify the L or E form of a macro-instruction.
- The value mnemonics shown in the format description specify the operand forms that are allowed in the standard form of the macro-instruction. Different operand forms are allowed in the L and E forms.
- The format description shows the combination of operands that is required to request the control program service. If a particular operand is required (i.e., it is not optional), it need be written in only one of the macro-instructions that comprise an L- and E-form pair. Furthermore, a particular operand can be written in either or both of the macro-instructions of the pair.

When there is deviation from these requirements, the special requirements for writing the operands are described in the "L- and E-Form Use" paragraph of the macro-instruction.

Typical special operand requirements are:

- A particular operand is not allowed in either the L or E form of the macro-instruction.
- A particular operand is required in either the L or E form of the macro-instruction, or both.
- In addition to the MF operand, there are operands that are not allowed in the standard form of the macro-instruction, but that are allowed or required in the L or E form.

For example, in the WTOR macro-instruction, the message operand is not allowed in the E form of the macro-instruction. The message operand cannot be written because the macro-definition does not allow the message text, which is assembled in the parameter list, to be overwritten.

Another example of special operand requirements is provided by the READ macro-instruction. The second operand of this macro-instruction actually serves as an adjunct to the mnemonic operation code; it informs the macro-definition of the type of READ operation specified by the macro-instruction. This operand must be written in both the L and E forms of the macro-instruction. (The first operand is the address of a decb, which is the parameter list resulting from the standard and L forms of the macro-instruction. The remote parameter list referred to by the E form of the macro-instruction is also referred to by the first operand, and, therefore, the E form is specified by writing only MF=E. Refer to Section 3 for more details.)

A final example is provided by the PARAM operand of the XCTL macro-instruction. This operand results in a parameter list that is referred to by the program given control by the macro-instruction. Because the program containing the XCTL can be destroyed after execution of the macro-instruction, the PARAM operand is allowed in the L and E forms of the macro-instruction but not in the standard form.

CONTENTS CONTROL

To achieve optimum performance when dynamic program management is used, the programmer should be familiar with the following terminology and rules for controlling the contents of main storage.

REUSABILITY

Each load module has a reusability attribute that is specified at linkage editor time. This attribute applies to the entire load module, even though the module may be given control through references to either a member name or one of several aliases. Reusability attributes are as follows:

- Not reusable. A module of this type modifies itself during its use, so that a new copy of the module is required each time the module is used.
- Serially reusable. A module of this type modifies itself during its use. However, because the module initializes itself each time at the beginning of its use, the same copy of the module can be used more than once. The module must be used in a serial manner; that is, one use of the module must be complete before the next one begins.
- Reenterable. A module of this type does not modify itself during its use. Therefore, the same copy of the module can be used more than once. The module can be used in a parallel manner; that is, one use of the module need not be complete before the next one begins. (The module can be used by one task in an iterative manner, and it can be used concurrently by two or more tasks.)

LIBRARIES

The modules used by a job step can be dynamically acquired from three sources (partitioned data sets). These sources are:

- Link library. This library is automatically available to all job steps. The necessary data control block is a permanent part of the control program and is always open.
- Job library. This library is available to a job step only if the JOBLIB DD statement was included in the job input stream. The necessary data control block is provided by and opened by the control program.
- Private library. One or more private libraries are available to a job step only if appropriate DD statements were included with the job step in the job input stream. The necessary data control blocks must be provided by and opened by the programs of the job step.

PACK AREAS

Problem program load modules are loaded into main storage in one of two areas, which are called pack areas (because modules are packed together within the areas).

The link pack area contains reenterable modules obtained from the link library. The link pack area is used by all tasks in the system, and has the control program's storage protection key.

The job pack area contains all other modules obtained for use by one job step. Each job step has its own job pack area, and this area has the job step's storage protection key. This area is freed for other uses when the job step terminates.

NOTE: If option 4 was excluded from the system, there is one pack area for each step; that is, the link pack area and the job pack area are combined for each step.

CONTENTS DIRECTORY

In the context of a particular partitioned data set, load modules are referred to by member names and aliases. These names are assigned at linkage edit time.

The member name of the load module is the name associated with the standard entry point of the module. Up to five additional names can be assigned as aliases to the module name. If the alias is the same as an entry point name within the module, execution of the module can begin at that point. If the alias is not the same as an entry point name, execution can begin at the standard entry point. The member name and the aliases are called entry point names and can be up to eight characters long.

When a load module is loaded into main storage, an entry point name is placed in a control program table, which is called the contents directory. The load module is acquired by searching for the entry point name in the directory of a specified partitioned data set.

When module loading results from a reference to a member name, the member name is placed in the contents directory. Aliases of the member name, if any, are not placed in the contents directory at this time. When module loading results from a reference to an alias, the single alias and the member name are placed in the contents directory. If a loaded module is referred to by an alias that is not in the contents directory, the directory of the partitioned data set must be searched for the alias; the member name associated with the new alias is determined, and it can then be determined that the module is already in main storage. The new alias is then placed in the contents directory.

Entry point names can also be placed in the contents directory by means of the IDENTIFY macro-instruction.

If option 4 was excluded from the system, an alias name is treated as a member name.

LOAD MODULE ACQUISITION PROCEDURES

When a load module is requested by means of the LOAD, LINK, XCTL, or ATTACH macro-instruction having the EP or EPLOC operand, the contents directory and the libraries are searched for the requested load module in the following order:

1. The contents directory entries for load modules contained in the job pack area are searched for an entry containing the specified entry point name.

2. If the DCB operand was written in the macro-instruction, the specified library is searched.

3. If the DCB operand was omitted in the macro-instruction, the job library, if it exists, is searched.

4. The contents directory entries for load modules contained in the link pack area are searched for an entry containing the specified entry point name.

5. The link library is searched.

The entry point name specified in the macro-instruction can be a member name or an alias, or it can be an entry name defined by using the IDENTIFY macro-instruction.

If the DE operand, instead of the EP or EPLOC operand, was written in the macro-instruction, the contents directory and the libraries are searched for the requested load module in the following order:

1. The contents directory entries for load modules contained in the job pack area are searched for an entry containing the entry-point name that is part of the specified list entry.

2. If the specified list entry is for a load module contained in the link library, the contents directory entries for load modules contained in the link pack area are searched.

3. If the DCB operand was written in the macro-instruction, the specified list entry is used to load the requested load module from the library designated by the specified data control block.

4. If the DCB operand was omitted in the macro-instruction, the specified list entry is used to load the requested load module from the job library or the link library, depending on which library is designated by the list entry.

The entry point name specified in the macro-instruction can be a member name or an alias.

If option 4 was excluded from the system, the contents directory entries for modules contained in the link pack area are searched at the same time as the entries for modules contained in the job pack area. If EP or EPLOC is written, neither the link library nor the job library is searched if the DCB operand is present. An entry point name defined by an IDENTIFY macro-instruction can be specified only in an ATTACH macro-instruction.

USE OF THE LOAD MACRO-INSTRUCTION

The LOAD macro-instruction is issued by a task to ensure that a specified load module will be in main storage until the task either issues a DELETE macro-instruction or terminates. During the time that the task requires the presence in storage of the load module, the task is said to be "responsible" for the module.

A module copy that satisfies the requirements of a LOAD issued by one task can, in certain cases, simultaneously satisfy the requirements of LOAD macro-instructions issued by other tasks. In this case, all of the tasks that issued the LOAD macro-instructions are responsible for the copy. The copy cannot be released from storage until each of the tasks relinquishes its responsibility.

The concept presented above is reflected in a responsibility count in the contents directory entry for a module. This count is incremented by one when an appropriate LOAD is executed, and it is decremented by one when an appropriate DELETE is executed.

In addition, a responsibility count is maintained for the module in internal control blocks associated with each task responsible for the module. (Note that one task can issue two or more LOAD macro-instructions for the same module without issuing intervening DELETE macro-instructions.) When a task terminates, its responsibility count for a module is subtracted from the responsibility count for the module in the contents directory entry. This ensures that the module will be released from storage when it is no longer required, even when the number of DELETE macro-instructions that were issued for the module is less than the number of LOAD macro-instructions that were issued.

The exact action of a LOAD macro-instruction varies according to the type of module that it specifies. The four possible module types are:

- Reenterable module from the link library.
- Reenterable module from a job library or private library.
- Serially reusable module from any library.
- Nonreusable module from any library.

REENTERABLE MODULE FROM THE LINK LIBRARY

The module is loaded into the link pack area. A single copy of the module can satisfy the requirements of all tasks in the system.

If option 4 was excluded from the system, separate copies are required for each job step.

REENTERABLE MODULE FROM A JOB LIBRARY OR PRIVATE LIBRARY

The module is loaded into the job pack area. A single copy of the module can satisfy the requirements of all tasks of the job step.

SERIALLY REUSABLE MODULE FROM ANY LIBRARY

The module is loaded into the job pack area. A single copy of the module can satisfy the requirements of all tasks of the job step. A LOAD will be satisfied by a module copy that is already in main storage, even though the module is currently being used in a type II linkage. The supervisor queues tasks that concurrently request use of the module in type II linkages. The ENQ and DEQ macro-instructions must be used if any task uses the module in a type I linkage. In this case, tasks that use the module in type II linkages must also use the ENQ and DEQ macro-instructions.

If option 4 was excluded from the system, a module that is already in main storage can satisfy a LOAD macro-instruction only if it was loaded by a previous LOAD macro-instruction within the same job step.

NONREUSABLE MODULE FROM ANY LIBRARY

The module is loaded into the job pack area. If option 4 has been excluded from the system, a single copy is used to satisy all the requirements of the job step; otherwise, a copy of the module is loaded each time a LOAD macro-instruction is executed. Each copy can be used by any task of the job step. Each copy can be released from storage only by execution of a DELETE macro-instruction by the task that caused the copy to be loaded, or by termination of this task. If the task that loaded the copy does not use it, but another task does, the task that loaded the copy still must release it. Release of the copy is delayed until the copy is no longer being used for a type II linkage.

Copies retained for use by a task (i.e., copies satisfying LOAD requests from the task) are released in first-in, first-out order. Completion of a type II linkage to a reserved module copy does not cause the copy to be released; a DELETE for the copy must still be issued. However, every type II linkage must be satisfied by an unused copy. If a retained unused copy is available when a type II linkage occurs, that copy will be used; if an unused copy is not available, a new copy will be loaded. In this latter case, upon completion of the type II linkage, the copy will be released; a DELETE is not required. If type I linkages are used, the programmer must provide protection against multiple uses of the same copy.

USE OF THE IDENTIFY MACRO-INSTRUCTION

The IDENTIFY macro-instruction is issued by a task to define an additional entry point either in the load module containing the IDENTIFY macro-instruction or in a load module that was loaded by a LOAD macro-instruction issued by the same task. The "identified" entry point name will be retained in the contents directory as long as the load module containing the entry point is in main storage.

If the "identified" entry point is referred to by a LINK, XCTL, or ATTACH macro-instruction, the program associated with the entry point is assumed to be reenterable. This occurs even when the load module that contains the entry point is known to be not reenterable.

The identified entry point can be referred to by the task issuing the IDENTIFY macro-instruction, and by any other task in the same job step. If the load module containing the entry point is in the link pack area, the entry point can be referred to by any task in the system.

If option 4 was excluded from the system, the identified entry point can be referred to only by the ATTACH macro-instruction.

USE OF THE LINK, XCTL, AND ATTACH MACRO-INSTRUCTIONS

Table 39 describes the program management provided by the supervisor in type II linkages.

Table 39. Program Management in Type II Linkages

Program Type	Action When LINK, XCTL, or ATTACH Is Executed
Reenterable from link library	A single copy is loaded into the link pack area for use by all tasks of all job steps.
Reenterable from job library or private library	A single copy is loaded into the job pack area for use by all tasks of the job step.
Serially reusable from any library	A single copy is loaded into the job pack area for use by all tasks of the job step. The supervisor queues tasks that require the program. If LOAD and type I linkages are also used, the ENQ and DEQ macro-instructions must be used.
Nonreusable from any library	Every linkage requires an unused copy. If LOAD and type I linkages are also used, the programmer must provide protection against multiple uses of the same copy.

If option 4 has been excluded from the system, a type II linkage normally causes a new copy of the requested module to be loaded into the appropriate pack area. A previously loaded copy is reused only if one of the following conditions is met:

- The linkage macro-instruction is ATTACH, and it specifies an entry point previously identified by an IDENTIFY macro-instruction.
- The requested module was loaded by a LOAD macro-instruction and is not currently being used in a type II linkage.
- The linkage macro-instruction is ATTACH or LINK; the requested module is the module most recently used in a type II linkage, is serially reusable or reenterable, and has not been destroyed since its last use.

As an option the user can specify the address of an exit list in his data control block. The entries in this list provide user control or provide control program action at specified times (or when certain situations arise). If the exit list option is chosen, the list must be constructed by the user, and the address of the list must be specified in the EXLST operand of the DCB macro-instruction or by the user's problem program before the pertinent macro-instruction is issued.

The exit list is constructed of four-byte entries that must be aligned on full-word boundaries. The contents of the entries are as shown in Table 40. The type of entry is specified by a code in the high-order byte; an address is specified in the three low-order bytes.

Table 40. Format and Contents of an Exit List

Type of Exit List Entry	Hexadecimal Code (high-order byte)	Contents of Exit List Entry (three low-order bytes)
Inactive entry	00	Ignored; the entry is not active.
Input header label	01	Address of user routine to process a user input header label.
Output header label	02	Address of user routine to create a user output header label.
Input trailer label	03	Address of user routine to process a user input trailer label.
Output trailer label	04	Address of user routine to create a user output trailer label.
Data control block exit	05	Address of the user's data control block exit routine.
Checkpoint following end of volume (at the beginning of the next volume)	06	Address of a user-supplied data control block to be used as the operand of a checkpoint (CHKPT) macro-instruction. The control program will issue the CHKPT macro-instruction at the specified time, supplying the address of the data control block stated here.
Last	80	Last entry in list. This code may accompany any of the above but must always accompany the last entry in the list.

The user may activate (or deactivate) any entry by placing the required code in the high-order byte. The list will be scanned downward; the first active entry encountered that has the proper code will be selected.

The user may shorten the list during execution by setting the high-order bit of the high-order byte of an entry to the value 1. This establishes that entry as the last in the list. Conversely, he can extend the list by setting the bit to zero. The user is cautioned not to accidently destroy the code for the last entry in the list when activating or deactivating entries.

The contents of the general registers when control is passed to either user label exit routines or data control block exit routines are as follows:

Register	Contents
0	For user label exits only, points to label buffer area.
1	Points to data control block currently being processed.
2 to 12	The contents that existed before the macro-instruction was executed.
13	Contents unchanged.
14	The return address (must not be altered or destroyed by user-written routine).
15	The address of the entry point to a user-written routine.

LABEL EXIT ROUTINES: Table 43 lists the label exits provided for BSAM and QSAM, and indicates the times at which the label exit routines are executed. Each exit routine must terminate with a RETURN macro-instruction and must provide a return code that indicates the action to be taken by the control program. This action also depends upon whether the exit routine is creating or returning labels, as shown in Table 41.

Table 41. Control Program Response to an Edit Routine Return Code

Return Code	Control Program Action
	Labels are being retrieved - Input Processing Routine
0	Specifies that normal processing should be resumed; any additional user labels will be ignored.
4	Specifies that another user label is to be read, and that control should again be given to the user label exit routine. If another user label does not exist, normal processing will be resumed.
	Labels are being created - Output Processing Routine
0	Specifies that the label created by the user is the last such label to be written. Control should not again be given to the user label exit routine after the current label is written.
4	Specifies that control should again be given to the user label exit routine after the current label is written, provided that the maximum number of labels (eight) has not been exceeded.

CAUTION: Register 14 contains a return address and must not be altered by the user written exit routine. If any control program operations are requested by the user while in the exit routine, the user must preserve the contents of register 14 and restore the contents before issuing the RETURN macro-instruction.

DATA CONTROL BLOCK EXIT ROUTINE: The data control block exit routine is executed during the opening process after the job file control block has been used to supply information to the data control block. Each data control block exit routine must terminate with a RETURN macro-instruction. In this case, no return code is required.

CAUTION: Refer to caution for label exits.

Exit List Example: Table 42 illustrates a sample exit list. The symbolic addresses are those of the exit routines.

Table 42. Exit List

Name	Operation	Operand	Comments
Symbol	DC	X'03'	Input user trailer label
	DC	AL3(ITLBL)	Routine address
	DC	X'01'	Input user header label
	DC	AL3(IHLBL)	Routine address
	DC	X'05'	Data control block exit
	DC	AL3(MYDCBXT)	Routine address
	DC	X'00'	Inactive
	DC	AL3(OHLBL2)	Routine address
	DC	X'02'	Output user header
	DC	AL3(OHLBL1)	Routine address
	DC	X'04'	Output user trailer label
	DC	AL3(OTLBL)	Routine address
	DC	X'86'	Checkpoint (06) at end of volume and last entry (80)
	DC	AL3(CKPT)	Address of the data control block to be used to record the checkpoint

The following two exit routine examples show the relationship of the exit list to the RETURN macro-instruction. OHLBL1 and MYDCBXT are entry points to the user's routines. The RETURN macro-instructions are used to return to the control program.

```
OHLBL1                  Entry point, output user header label routine
          .
          .
          .
          BH        EEQ
          .
          .
          RETURN    RC=0    Return to control program
EEQ       RETURN    RC=4    Return to control program
MYDCBXT   .                 Entry point, data control block exit routine
          .
          .
          .
          RETURN    ,       Return to control program
```

LABEL EXITS: Table 43 lists the label exits provided for BSAM and QSAM, and the times at which they are taken. The situations describing end of volumes include end of data sets for input data sets.

Table 43. Label Exits

Opt[1] of OPEN	Exit Situations	Exits Associated With			
		Old Volume (user trailer labels)		New Volume (user header labels)	
		Input Exit 3	Output Exit 4	Input Exit 1	Output Exit 2
INPUT or UPDAT or INOUT (no writes)	User issues OPEN			X	
	User issues FEOV			X[1]	
	At end of volume	X		X	
	At end of data set	X[2]			
OUTPUT or OUTIN or INOUT (one or more writes)	User issues OPEN				X
	User issues CLOSE		X		
	User issues FEOV		X		X
	At end of volume		X		X
		(user header labels)		(user trailer labels)	
RDBACK	User issues OPEN			X	
	User issues FEOV			X[1]	
	At end of volume	X		X	
	At end of data set	X[2]			

[1] Exit taken if not at end of data set
[2] Exit taken before user's end-of-data set routine is entered.

The System/360 Operating System uses several groups of standard labels to identify tape and direct-access volumes and the data sets they contain on secondary storage. The labels are used to locate the data sets and are identified and verified by the label processing routines.

STANDARD MAGNETIC TAPE LABELS

All standard tape labels are 80 character records. They are written in extended binary coded decimal interchange code (EBCDIC) on nine track tape units and in binary coded decimal (BCD) on seven track tape units. In binary coded decimal, even parity is used. The tape label is recorded in the same density as the data on the tape.

The following label groups are included:

- Volume label group.
- Data set header label group.
- User header label group (optional).
- Data set trailer label group.
- User trailer label group (optional).

VOLUME LABEL GROUP

A volume label group is composed of an initial volume label and up to seven additional volume labels. The initial volume label identifies a volume and its owner, and is used to verify that the correct volume is mounted. It can also be used to prevent use of the volume by unauthorized programs.

Additional volume labels are processed by an installation supplied routine.

Initial Tape Volume Label Format

Field	1	2	3	4	5	6	7	8	9
Position	1	3	4	5	10	11	21	31	41	51	80
Length	3	1	6	1	10	10	10	10	29

Field #1 - Label Identifier. The contents of this field, the characters VOL, indicate that this is a volume label.

Field #2 - Label Number. This field identifies the relative position of the volume label in a volume label group. It is written as 1.

Field #3 - Volume Serial Number. This field contains a unique identification code. The identification code is assigned when the volume enters the system. It can be placed on the external surface of the volume for visual identification. The field is normally numeric (000001-999999), but may be any six alphameric characters.

Field #4 - Volume Security. This field is reserved for future use by installations that wishes to provide security at the volume level.

Field #5 - Data Set Directory. This field is currently not used for tape volume labels. It will be blank.

Field #6 - Reserved for Manufacturers. This field is reserved for future standardization purposes. It will be blank.

Field #7 - Reserved. This field is reserved for future developmental purposes. It will be blank.

Field #8 - Owner Name and Address Code. This field contains a unique identification of the owner of the volume.

Field #9 - Reserved. This field is reserved for future developmental purposes. It will be blank.

Additional Volume Labels Format

Field	1	2	3
Position	1 3	4	5 80
Length	3	1	76

Field #1 - Label Identifier. The contents of this field, the characters VOL, indicate that this is a volume label.

Field #2 - Label Number. This field identifies the relative position (2 to 8) of the volume label in the volume label group.

Field #3 - User Specified.

DATA SET HEADER LABEL GROUP

The data set header label group consists of two labels (HDR1 and HDR2) written in the format shown below.

Data Set Header 1 Label Format

Field	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Location	1 3	4	5 21	22 27	28 31	32 33	34 35	36 39	40 41	42 47	48 53	54 54	55 60	61 80
Length	3	1	17	6	4	4	4	2	6	6	1	6	13	7

Field #1 - Label Identifer. The contents of this field, the characters HDR, indicate that this is a data set header label.

Field #2 - Label Number. This field identifies the relative position of the data set label in the data set label group. It is written as 1.

Field #3 - Data Set Identifier. This field identifies the data set.

Field #4 - Data Set Serial Number. This field contains the same identification code that appears in field #3 of the initial volume label of the only, or first, volume of the data set or multi-data set aggregate.

Field #5 - Volume Sequence Number. This field indicates the volume on which the data set is recorded in relation to the volume on which the data set begins.

Field #6 - Data Set Sequence Number. This field indicates the position of the data set in relation to the first data set in the aggregate.

Field #7 - Generation Number. This field indicates the generation number (0000-9999) of the data set.

Field #8 - Version Number of Generation. This field indicates the version of a generation of the data set.

Field #9 - Creation Date. The contents of this field indicate the year and the day of the year the data set was created. It is recorded as bYYDDD, where YY is 00-99 and DDD is 001-366.

Field #10 - Expiration Date. The contents of this field indicate the first day the tape may be overwritten. It is recorded as bYYDDD.

Field #11 - Data Set Security Indicator. This field indicates whether additional qualifications must be supplied in order to process the data set. A 0 indicates that no additional qualifications are required. A 1 indicates that additional qualifications are required.

Field #12 - Unused. This field contains zeros.

Field #13 - System Code. This field contains a unique identification of the programming system.

Field #14 - Reserved. This field is reserved for future standardization. It is blank.

Data Set Header 2 Label Format

Field	1	2	3	4	5	6	7	8	9
Position	1	3	4	5	6	10	11	15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
Length	3	1	1	5	5	1	1	17	46

Field #1 - Label Identifier. The contents of this field, the characters HDR, indicate that this is a data set header label.

Field #2 - Data Set Label Number. This field identifies the relative position of the data set label in the data set label group. It is written as 2.

Field #3 - Record Format. This field indicates the format of the records as follows:

F - fixed length
V - variable length
U - undefined length

Field #4 - Record Length. This field indicates the length of the records: interpretation of this field depends on the format specified in field #3, as follows:

Form F - block length
Form V - maximum block length
Form U - five high-order digits of a ten-digit field (fields 4 and 5 denote maximum block length)

Field #5 - Blocking Factor/Block Size. This field indicates the blocking factor or block size, depending on the format specified in field #3, as follows:

Form F - blocking factor
Form V - maximum logical record length
Form U - five low-order digits denoting the maximum block length

Field #6 - Density. This field indicates the tape density as shown in Table 44.

Table 44. DEN Values

DEN Value	Tape Recording Density (bits/inch)		
	Model 2400		Model 7340
	7 Track	9 Track	
0	200	-	1511
1	556	-	3022
2	800	800	-

Field #7 - Data Set Position. This field indicates whether a volume switch has previously occurred for the data set. If a volume switch has occurred, 1 is written. If no volume switch has occurred, 0 is written. When the tape is read backwards, this information indicates when a volume switch is required.

Field #8 - Job/Job Step ID. This field indicates the job and job step identification that was assigned by the task scheduler. It provides specific time-dependent identification of the data set created by the operating system.

Field #9 - Reserved. This field is reserved for future requirements. It is blank.

USER HEADER LABEL GROUP

Up to eight user header labels can follow the data set header label group. The labels are written by the operating system as directed by the problem program that records the data set.

When the data set is retrieved, the user header label group is made available to the problem program by the operating system.

User Header Label Format

Field	1	2	3
Position	1 3	4	5 ... 80
Length	3	1	76

Field #1 - Label Identifier. The contents of this field, the characters UHL, indicate that this is a user header label.

Field #2 - Label Number. This field identifies the relative position (1 to 8) of the label within the user label group.

Field #3 - User Specified.

DATA SET TRAILER LABEL GROUP

The data set trailer label group consists of two labels written in a format identical to the data set header labels except for the following fields:

Field #1 - Label Identifier. The contents of this field indicate that this is a data set trailer label and marks an end of volume (EOV) or an end of data set (EOF).

Field #2 - Label Number. This field indicates that the label is the first (1) or second (2) data set trailer label.

Field #12 - Block Count. This field of the data set trailer 1 label indicates the number of blocks on the data set or on the current volume of a multivolume data set.

USER TRAILER LABEL GROUP

Up to eight user trailer labels can follow the data set trailer label group. Processing of user trailer labels is as described for user header labels. They are written in a format identical to the user header labels except for field #1, which is written as UTL.

DIRECT-ACCESS VOLUME LABELS

All direct-access volume labels are written in extended binary-coded-decimal interchange code (EBCDIC). The following label groups are written:

- Volume label group.
- Data set control block group.
- User header label group (optional).
- User trailer label group (optional).

VOLUME LABEL GROUP

A volume label group is composed of an initial volume label and up to seven additional volume labels. The initial volume label identifies a volume and its owner, and is used to verify that the correct volume is mounted. It can also be used to prevent use of the volume by unauthorized programs.

Additional volume labels are processed by an installation-supplied routine.

The format of the direct-access volume label group is the same as the format of the tape volume label group, except for field #5 of the initial volume label. This field contains the address of the volume table of contents (VTOC) of the direct-access volume.

DATA SET CONTROL BLOCK GROUP

The data set label of a data set on a direct-access volume consists of the data set control block (DSCB). The DSCB appears in the volume table of contents (VTOC) and contains the equivalent of the tape data set header and trailer information, in addition to space allocation and other control information.

USER HEADER AND TRAILER LABEL GROUPS

Up to eight user header labels and eight user trailer labels may appear on a single track, which is allocated for this purpose when the data set is created. The processing and format of these labels is the same as described for tape user labels.

CONTROL CHARACTERS

As an optional feature, all record formats may include a control character in each logical record. This control character will be recognized and processed by QSAM and BSAM if a data set is being written to a printer or punch.

For format-F and -U records this character is the first byte of the logical record.

For format-V records it must be the fifth byte of the logical record, immediately following the logical record length field.

Two options are available. If either option is specified in the data control block, the character must appear in every record and other line spacing or stacker selection options also specified in the data control block are ignored.

MACHINE CODE

The user can specify in the data control block that the machine code control character has been placed in each logical record. If the record is to be written, a byte supplied by the user must contain the command code bit configuration specifying both the write and the desired carriage or stacker select operation. If the record is not to be written, the byte supplied by the user can specify any command other than write.

Command codes for specific devices are contained in IBM System Reference Library publications describing the control units or devices.

EXTENDED ASA CODE

The user may choose to specify this code rather than the machine code. For punch operations, the record is always written.

The code is as follows:

Code	Interpretation
SP	Space one line before printing (blank code)
0	Space two lines before printing
-	Space three lines before printing
+	Suppress space before printing
1	Skip to channel 1
2	Skip to channel 2
3	Skip to channel 3
4	Skip to channel 4
5	Skip to channel 5
6	Skip to channel 6
7	Skip to channel 7

8	Skip to channel 8
9	Skip to channel 9
A	Skip to channel 10
B	Skip to channel 11
C	Skip to channel 12
V	Select punch pocket 1
W	Select punch pocket 2

These codes include those defined by ASA FORTRAN. If any other code is specified, it is interpreted as SP or V, depending on the device being used; no error indication is returned to the user.

SYSOUT WRITERS

Data definition statements defining output data sets can specify SYSOUT processing. The functions provided by SYSOUT are discussed in the publication IBM System/360 Operating System: Operator's Guide, Form C28-6540.

SYSOUT data sets are usually intended for unit-record devices and must have sequential organization. The programmer uses the OPEN and CLOSE macro-instructions in the usual fashion, specifying either the basic sequential access method (BSAM) or the queued sequential access method (QSAM). In systems without output writers, problem programs usually write SYSOUT data sets directly to unit-record devices. The type of device involved for any particular execution of a problem program is determined by the DD statement that defines the data set for that execution. To ensure that a data set will always be eligible for SYSOUT disposition, the user should open it in a device-independent manner, so that a DD statement will have the freedom to specify the appropriate device type. This requires that the DEVD operand in the DCB macro-instruction reserve the largest device-dependent area.

The logical records can be written in any format defined for the particular combination of access method and device type involved. These definitions are discussed under "Access Methods for Sequential Data Sets" in the publication IBM System/360 Operating System: Data Management. In all systems, the logical record length must never exceed the maximum allowable for the unit-record device on which the data set will ultimately be written. Output writers punch only the EBCDIC mode.

The problem program is responsible for any formats, pagination, or header control. Control character usage is specified in the usual manner by the user (DCB macro-instruction or alternate source). If no control characters are used, in systems having output writers, a standard control is supplied; with printers, for example, the output writers will single-space and skip to channel 1 when channel 12 is sensed, while for punches stacker 1 is selected.

The standard status indicators provided on entry to the SYNAD routine are arranged in main storage as follows:

Location + 2 bytes		sense byte 1
+ 3 bytes		sense byte 2
+ 8 bytes		first byte of channel status word (CSW)
+ 9 bytes	(CSW)	command address
+ 12 bytes	(CSW)	status bytes 1 (unit)
+ 13 bytes	(CSW)	status bytes 2 (channel)
+ 14 bytes	(CSW)	count field

The address, location, is provided either in the data event control block or a register.

The first six bits of the first sense byte and the two status bytes are device independent and are shown below. (The additional bits of sense byte one and all of sense byte two are device dependent, and the individual device manuals should be referred to.)

Note: Sense byte 1 and 2 are meaningful only when bit 6 (unit check) of status byte 1 is set.

Sense (byte 1)	Status (byte 1)	Status (byte 2)
Bit	Bit	Bit
0 Command reject	0 Attention	0 Program-controlled interruption
1 Intervention required	1 Status modifier	1 Incorrect length
2 Bus out check	2 Control unit end	2 Program check
3 Equipment check	3 Busy	3 Protection check
4 Data check	4 channel end	4 Channel data check
5 Over run	5 Device end	5 Channel control check
	6 Unit check	6 Interface control check
	7 Unit exception	7 Chaining check

ABEND macro-instruction
  action with ECB operand 91
  and abnormal termination dump 103
  and post codes 91
  description 102
  (also see subtask termination; STAE macro-instruction)
Absexp (see absolute expression)
Absolute expression
  defined 304
  examples of 304
  represented by absexp 18
  with implied and explicit addresses 304-306
  with register notation 306
  (also see L- and E- forms of macro-instructions)
Action
  function of TESTRAN macro-instructions 243
  operand of CNTRL macro-instruction 178
Action macro-instructions (TESTRAN) 247-257
Actual addresses
  as feedback from BDAM 229-231
  defined as MBBCCHHR 115
  result in unmovable data sets 115,184
  with QISAM
    provided in load-mode 203
    used in scan-mode 208
Addr
  notation defined 17
  operand forms 303
  operand processing 309,310
  used with system macro-instruction 18
  (also see expression; register notation; L- and E- forms of macro-instructions)
Address constants
  A-type
    in overlay management 52
    resulting from operand processing 310
  V-type
    in overlay management 50,51
Addresses
  as output from TESTRAN TRACE FLOW 285,288,289,290,292
  as parameters 12
  assigned to overlay segments 48
  at which SVC instructions are inserted by TESTRAN 244,260
  explicit 304,305
  implied 304
  of search arguments in BDAM 231
  of search arguments in QISAM 208
  passed in standard registers 27
  placed in parameter lists 12
  relative block 116,193,229,231,236
  relative track 116,173,193,231,232,236
  track 115
  when scatter loading with TESTRAN 249,255
  (also see mnemonic; L- and E- forms of macro-instructions; and V-type address constants)
Addrx
  notation defined 17
  operand forms 303
  operand processing 309,310
  used to same effect as register notation 306,313
  used with system macro-instructions 18
Addx
  notation defined 17
  operand forms 303
  used with TESTRAN macro-instructions 245
Adval
  notation defined 17
  used with TESTRAN macro-instruction 245
Alias
  as an entry point name 60,318,319
  in a partitioned organization directory 184
  multiple aliases 318
  when processing a partitioned data set 192,193
Allocating
  buffer pools
    GETBUF macro-instruction 130
    GETPOOL macro-instruction 127
  storage
    GETMAIN macro-instruction 65,67
  subpools
    ATTACH macro-instruction 77
  tracks for BDAM data set 181
Area
  cylinder overflow 195
  independent overflow 195
  pack 318-322
  save
    chaining 32-34
    contents 30-31
    use of 28,29,35,45,46
  (also see buffering)
ASA code, extended
  (see codes)
Assembler language
  as used by called program 35
  as used by overlay program 49,50
  CSECT statement in TESTRAN 244,247
  ENTRY statement 45,49
  expressions, review of 303-306
  SVC instruction 244
  table used by TESTRAN editor 244,276,300
Asynchronous exits
  and ABEND macro-instruction 104
  and PRTOV macro-instruction 154,177
  and RETURN macro-instruction 46
  and STIMER macro-instruction 108
  defined 26
  specified in SPIE macro-instruction 97
  to a called program 34

Asynchronous operation
and TESTRAN 254
and WAIT macro-instruction 89
influence of CHAP macro instruction 85
initiated by
ATTACH macro-instruction 75,80
SEGLD macro instruction 50
restrictions requiring the use of ENQ and DEQ macro-instructions 94
with event control block 77,91
ATTACH macro-instruction
and address of subtask's TCB 80
and load module attributes 322
and use of private libraries 76
contention for CPU 80
description 75
effect of EP or EPLOC operands 319
effect of specifying ECB operand 48,80,91
restriction with TESTRAN 254
(also see asynchronous operations)
Attributes
for TESTRAN data editing 18,244,250,252,263,308
of data
implicit 252
in assembler symbol table 244
notation 308,309
overriding 249
scale 18,244,250,252,263,308,309
type 244,252,263,308
of data control block name fields 119
of data sets 184
of load modules 54,78,317

Backward reading
(see READ macro-instruction)
BDAM (basic direct access method) 224-238
BISAM (basic indexed sequential access method) 213-224
Blank
as a delimiter 36
as ASA code 333
with TESTRAN editor 301
BLDL macro-instruction
description 190
use with ATTACH macro-instruction 76
use with STOW macro-instruction 193
Blocks, control
(see buffer pool control block; data event control block; data extent block; DCB macro-instruction; event control block; queue control block; and task control block)
Blocks, data
address in secondary storage 115,116
count 331
identification 173
input of
using BDAM 229-230
using BSAM 164-165
length defined in label 330,331
output of
using BDAM 233-237
using BSAM 166-167,179-182
updating of
using BSAM 167-168

Braces
defined 20
Brackets
defined 20
Branch instruction
as linkage 25,26,28
using return codes 28
with overlay structure 49,50
(also see L- and E- forms of macro-instructions)
BPAM (basic partitioned access method) 182-194
BSAM (basic sequential access method) 155-182,334
BSP macro-instruction 176
Buffer (buffering)
Control program provides address
BDAM 230
BISAM 219
QISAM 209
QSAM 141,144,147
dynamic options
in BDAM 234
in BISAM 222
lengths
required in BISAM 221,222
required in QISAM 199,206,207
with chained scheduling 144
obtained by
BUILD macro-instruction 129
dynamic options in BDAM 225,230,234
dynamic options in BISAM 214,219,222
GETBUF macro-instruction 130
GETPOOL macro-instruction 127
released by
FREEBUF macro-instruction 131
FREEDBUF macro-instruction 223,238
FREEPOOL macro-instruction 128
required DCB macro-instruction operands
BDAM 228
BISAM 217
BPAM 162
BSAM 162
QISAM 198
QSAM 139
reuse of BUILD macro-instruction 129
Buffer acquisition
(see buffer (buffering))
Buffer pool control block 118,128
BUILD macro-instruction 129
(also see buffer (buffering))

CALL macro-instruction
and entry point identifier 34,49
and LOAD macro-instruction 62
and XCTL macro-instruction 60
description 41
in overlay management 48,49
in passing control information 36
Called program
as a lower control level program 28
conventions to be followed 34,35
register contents when receives control 28,29
save area chaining examples 32,33
save area requirements 30,31
(also see SAVE and RETURN macro-instructions)

Calling program
as a higher control level program 28
conventions to be followed 35
passing control information 36
register contents before relinquishing control 28,29
save area requirements 30,31
(also see CALL macro-instruction)
Capacity
of direct-access device track 114,115
record, defined 115
Card (read punch feed)
and DEVD operand
BSAM 159
QSAM 137
and RECFM operand
BSAM 160
QSAM 138
CNTRL macro-instruction 155,178
control characters listed 333
SYSOUT writers 334
Catalog 116
Cataloged procedure (see procedure)
Chaining
of input/output operations 142,144,336
of save areas 32
CHANGE control statement
and TESTRAN output 249
Channel
address in MBBCCHHR 115
skip in response to CNTRL macro-instruction 153,177
skip in response to control characters 334
status word 143,171,335
test for printer overflow 154,176
CHAP macro-instruction 85
Character
constants 18,308,309
control, unit record equipment 333
EOR delimiter 161
Check
chaining 336
channel control 335
data 335
interface control 335
program 254,335
unit 335
CHECK macro-instruction 169,171,174,177
Checkpoints 105-106
CHKPT macro-instruction 105
CLOSE macro-instruction
description 124
special use with BSAM 171
tape positioning after execution 125,172
Closing, parallel 172
CNOP instruction 13
CNTRL macro-instruction 155,177
(also see card)
Code
condition
compared to control statement 48
returned in BDAM 232
returned in BISAM 219
returned in QISAM 201
extended ASA 333
in CLOSE macro-expansion 124,126,172
in exit list 323
in OPEN macro-expansion 123
machine 333
operand of
CNTRL macro-instruction 155,177
DCB macro-instruction 136,158
POST macro-instruction 93
SETL macro-instruction 208
post 48,91,93
return 28,35,48,54,91,324
task completion 48
to specify TESTRAN editing 250,252,255,260,301,308
to specify TESTRAN service 245,316
Coded values 16,18,19,303
in TESTRAN 245
Comma
delimiter 14,36,307
Communications
between tasks
(see ATTACH macro-instruction POST macro-instruction; and RETURN macro-instruction)
by return codes
(see code)
using registers
(see registers)
Completion flag 91
(also see code)
Concatenation number 190,191
COND parameter
(see EXEC and JOB statements)
Condition
assigned to TESTRAN flag 268
error
(see data event control block; exit)
overflow
(see PRTOV macro-instruction)
wait 94,95
Console 39,308,314
Constants
characters 308,309
(also see address constants)
Content directory 318-320
Control, contents 317
Control area, program interruption (PICA) 35,98-101
(also see asynchronous exits)
Control blocks
(see data event control block; DCB macro-instruction; event control block; and task control block)
Control program
functions of 38
options for 39-40
Control section
with overlay 49
with TESTRAN 244,247
Conventions
for coding operands 20
for dummy control section 119
for linkages 34,35
Count
field in status indicators 335
in TESTRAN editing 301
of control information passed 36
responsibility, for a module 320
(also see block)

Count operand
as a NOP 92
of WAIT macro-instruction 89
of WAITR macro-instruction 93
Counter
location 260,304,305,306
program-testing 269
Creating
a direct organization data set 155
a sequential organization data set 148
an indexed sequential data set 194
labels, exits when 324
master indexes 195
Creating and attaching a task 75
CSECT instruction
(see control section)

Data, test 243
editing 243,244,249,250
identifying numbers for 247
recorded 250,252,254
Data chaining
(see chaining)
Data control block (DCB)
(see DCB macro-instruction)
Data definition statement
(see DD statement)
Data event control block (DECB)
format in BDAM 229
format in BISAM 218,219
format in BSAM 164
moving address of BDAM key field 235
with L- and E-forms of
macro-instructions 316
Data extent block 243
Data sets
concatenation of 190
partitioned
creation and accessing of 182-194
directory for 60,76,190
use of 317,318
(also see libraries; member)
Date
from TIME macro-instruction 107
in label fields 329
DCB macro-instruction
general description 117,118
in BDAM 225
in BISAM 214
in BPAM 186
in BSAM 156
in QISAM 194,204
in QSAM 134
in QTAM 239
modification of control block 119
DCB operand
in ATTACH 76
in LINK 53,55
in LOAD 61
in XCTL 58
DCBD macro-instruction 119,120
DD statement 117,124
Decimal integers 250,285,301,307,308,327
DELETE macro-instruction
description 62
used with LOAD macro-instruction
60,320,321
Deletion code 195,210,212
Density, tape 158
DEQ macro-instruction
description 96
used with ENQ macro-instruction
94,95,321
Description record, track 115
DETACH macro-instruction
description 84
used with ATTACH macro-instruction 80
used with LOAD macro-instruction
320,321
Diagnostic message
(see messages)
Direct-access
devices
actual address 115
control unit 223
relative address 116
work queue 39
restrictions with BSP macro-instruction
176
volumes
labels 327,332
positioning 123,171
Directory
contents 318-321
for work queue 39
of a partitioned data set 60,76,190
(also see alias; BLDL macro-instruction;
and IDENTIFY macro-instruction)
DUMP macro-instruction
DUMP CHANGES
description 249
edited 250
limit 275
output format 277
truncation 249
DUMP COMMENT
description 253
output format 281
DUMP DATA
description 247
output format 277
DUMP MAP
description 251
output format 279
DUMP PANEL
description 252
output format 280
DUMP TABLE
description 251
output format 279
Dynamic
testing 243,300
(also see buffer; and dynamic program
management)
Dynamic program management 38,317

Edit
errors found during 294
required statements 300,301
restrictions 275
test data 243,244,250,298
(also see editor)
Editor
linkage 48,300,301,317
TESTRAN 243,244,249,300,301

Elements
program interruption (PIE) 99-101
queue (QE) 94-96
Ellipsis 20
Embedded entry point 63
END operand 250,254,255
End of task 210,260
End-of-data-set exit
(see exit)
End-of-volume
condition 116,171
indication 331
routine 116,117,123
(also see exit)
ENQ macro-instruction 94,95
use with library module 321
Entry
equivalence of name and identifier 45
linkage 26
operand 62
point 26,28,49,54,60,62,318,321
point identifier 34,45
point name 49,60,62,318
point register 28
statement 45,49
(also see alias)
Equal sign 36
Error analysis
after CLOSE macro-instruction 125
after OPEN macro-instruction 124
BDAM 232
BISAM 219
BPAM 170,190,193
BSAM 170,171,174
QISAM 201
QSAM 141,142,146
standard status indicators 335
TESTRAN 243,268,269,275
Error messages 275,292,294
Error recovery
using a GO BACK macro-instruction 268
using a SET VARIABLE macro-instruction 269
ESETL macro-instruction 209
ETXR operand 77
(also see exit)
Even parity 158,327
Event control block (ECB)
effect upon subtask termination 80
format 91
operand in ATTACH macro-instruction 48,77
use in BDAM 229
use in BISAM 218
(also see ABEND macro-instruction; ETXR operand; and POST macro-instruction)
Exceptional condition handling 38,100
(also see error analysis; error recovery)
EXEC statement 36,48,300,301
Execution
of a BAL or LPSW instruction 26
of a calling sequence 35
of L- and E-form macro-instructions 312
of TESTRAN 307
and the assembler program 300
of an inserted SVC instruction 244
time conserved 243
resumption of
after LINK macro-instruction 54
after RETURN macro-instruction 46
after WAIT macro-instruction 91
after XCTL macro-instruction 36,316
(also see communications)
Exit
and ABEND macro-instruction
and overlay segments 49
conventions 45
DCB 117,119,123,325
end of data set (EODAD) 118,119,169
end of volume 116
EXTR 45
label 123,171,324,326
linkage 26
list
construction 323
sample 325
specify in SPIE macro-instruction 100
specify in STAE macro-instruction 45
specify in STIMER macro-instruction 45
system routine 100
(also see asynchronous; synchronous)
Expanding macro-instructions 11,12
Explicit address 305,306
Expressions
absolute 306
relocatable 304
Extended binary coded decimal interchange code (EBCDIC) 327,332
Extended search option 230,234
External
name 49
storage 243
symbol 50,249,263,305
symbol dictionary (ESD) 301
EXTR operand 77,80
(also see ABEND macro-instruction; asynchronous exit)
EXTRACT macro-instruction 87-88
EXTRN statement 304,305

Feedback
from NOTE macro-instruction 173
with BDAM 230,231,234,236
with QISAM 195,208
FEOV macro-instruction 127
FIND macro-instruction 189
FLAG
completion 93
logical, in TESTRAN 263,268,269
wait 91
(also see SET macro-instruction)
Floating-point registers 35,252,307
Format
-F records 152,158,161,164,171,187,231,235,236,333
-U records 142,146,158,161,171,187,235,333
-V records 161,235,333
of build list 190
of data event control block
BDAM 229
BISAM 218
BSAM 164
of event control block 91
of exit list 325

of labels 331,332
of macro-instructions 13-21
of program interruption control area 99
of program interruption element 99
of queue control block 95
of test data 243-245,249
of tracks 114
FREEBUF macro-instruction 131
FREEDBUF macro-instruction
described for BDAM 238
usage 230,236
described for BISAM 223
usage 213,221
FREEMAIN macro-instruction 71,72,83
FREEPOOL macro-instruction 128

GET macro-instruction
for QISAM
locate mode 209
move mode 210
for QSAM
locate mode 141
move mode 143
substitute mode 144
for QTAM 241
GETBUF macro-instruction 130
GETMAIN macro-instruction
described R-form 65
described S-form 67
used with ATTACH macro-instruction 81
GETPOOL macro-instruction 127
GO macro-instruction
GO BACK 244,267
to alter program flow 268
GO IN 266
GO OUT 267
GO TO 265

Heading, standard page 275,277
High-order bits
as end of variable list 36,223,224
in CLOSE parameter list 126,172
in FREEMAIN macro-instruction 71
in GETMAIN macro-instruction 67
in OPEN parameter list 123
in program interruption control area 99
STOW macro-instruction 194
use in register notation 18
Hyphen 17

Id
block field 115
label field 331
macro 247,294,298
operand 44,54
(also see CALL macro-instruction; entry;)
Identification
of data blocks
BDAM 229,231
BPAM 193
BSAM 173,174
of job and job step 331
of macro-instruction 290,294
of programming system 329
of volume owner 328
Identifier
data set 328
label 327,330,331
(also see CALL macro-instruction; entry; and id)
Identify macro-instruction
description 63
usage 54,60,318,321
Implied
address 304,306
attributes 276,308,309
volume positions 171
(also see addrx)
Inclusive
branch 50
segments 50
Indexed
implied address 307,314
sequential access method
(see BISAM; QISAM)
Indexes for data sets
levels of 197
Indicators
end-of-data 116,235
standard status 164,169,171,335
(also see exit; synchronous)
Information, standard status 143,335
Integer 17
TESTRAN register notation 307
TESTRAN value mnemonics 245
Interface
input/output status indicator 335
linkage responsibilities 34,35
Interpreter
see TESTRAN 243,244,260,263,265,267
Interruption
control area 35,99,100
element 99,100
masks 100
requested through
SPIE macro-instruction 97
STIMER macro-instruction 108
SVC 25,26,244
TESTRAN 244,260
Interval
counter 263
timer 108,111

Job
control statements
EXEC 36,48,300
JOB 48
file control block 325
identification 331
library 55,62,76,190,317,319,320
pack area 318,319,320
scheduler 46,48
scheduling 39
step 36,39,46,48,60,93,99,243,244,277,300,317
stream 32,39

Keyword operand 36,303
(also see L- and E-forms of macro-instruction)

L instruction
(see load instructions)
L- and E-forms of macro-instructions
described 312-316

MF operands 313
restrictions on value mnemonics 315,316
SF operands
in ATTACH macro-instruction 83-84
in LINK macro-instruction 55-56
in XCTL macro-instruction 59-60
LA instruction (see load instructions)
Labels, non-standard 173,174
Labels, standard
data set control block group 332
data set header 328-331
data set trailer 331
direct-access volume 332
formats 327-332
in end-of-data-set determination 116
user header 331,332
user trailer 332
volume 327-329,332
Languages
assembler 303-311
expressions in macro-instructions 303
in linkages 35
(also see instructions by name)
job control
to pass parameters to a program 36
use in jobs containing TESTRAN testing 300
(also see statements by name)
Length
block 171,330
of BISAM area 221
of buffers 118,199,205,216,227
of comment specified by TESTRAN macro-instruction 308
of key 187,197
of message specified by supervisor macro-instruction 308
of parameter lists 54,76
of record zero in standard track 115
of records 330,334
(also see control blocks, data)
overriding 219,223
wrong length indication 161
(also see attributes)
Level, control of programs 28,32-34
affect on RETURN macro-instruction 46,48
Level, index
(see indexes for data sets)
Libraries 48,76,317-321
job 317,319,320
link 317,318,319,320
private 317,319,320
use in dynamic program management 317
Line
output, in TESTRAN traces 290,292
printer 177,178,308,333
Link library
(see libraries)
LINK macro-instruction 52-56
in connection with traces 254
to specify calling sequence identifier 34
use of 26,32,33
use of in dynamic program management 322
Linkage editor
affect of CALL macro-instruction on linkage editing 62
job control statements for 300
only loadable modules 54
overlay programs 48-52
specification of reusability attributes 317
TEST option in EXEC statement for 301
use in TESTRAN testing 243,244,249
Linkage registers
(see registers)
Linkages
affect on TESTRAN traces 254
conventions 24
direct 244
interfaces for 34-36
macro-instructions for 38
registers for 27-28
return 26,32,50,141
supervisor-assisted 26,28,38,244
terminology 24
type I, II, III and IV 25-26,28,34-36,244,321,322
uses of 60,244,321-322
List, build 190
List, exit 118,123,323-326
List, parameter
fixed length 54,76
formed from sublists 76
in data event control block 164,218,229
location of 36,55
registers for 27,28,35,306
remote 60,309-316
types
problem program 28,55,60,76,306,310,311
supervisor 28,55,60
variable length 36,54,76
List entry 53,57,61,62,76,319
Literal 263,304,306
notation 309,310
pool 35
Load instruction (load (L), load address (LA), and load register (LR))
in L- and E-form macro-expansions 312
in macro-expansions 28,307,310,311
to set-up special register notation 143,148
used with explicit addresses 305
used with implied addresses 305
LOAD macro-instruction 60-62
in connection with CALL macro-instruction 62
in connection with DELETE macro-instruction 62-63
in connection with LINK macro-instruction 54,322
use of in dynamic program management 320-321
Load module
(see module)
Load program status word (LPSW) instruction
in synchronous exit 26
Loading of modules
as affected by priority 80
asynchronous 75-84
scatter 249,255
synchronous 52-62

Lower-case letters 17
LR instruction
(see load instructions)

Machine language
(see codes)
MACRF operand
Macro-definition 316
Macro-expansion
contents of, for LINK macro-instruction 54,55
of L- and E-form macro-instructions 312,313
of supervisor and data management macro-instructions 11
of TESTRAN macro-instructions 12,244,245
Macro-instructions
description of 37
format of 13-21
types of
R-type 21,307,310,311,313
S-type 22,307,309-315
(also see by name)
Magnetic tape
(see tape)
Main storage
allocation 65,67
conservation with
overlay 48
reenterable coding 318
TESTRAN 259
management 65-74
pack areas 318
release 71,72
references to 249,273
TESTRAN
map of 271
range of dump 247
range of trace 253,255,256
Mark, tape 116,176,178
MBBCCHHR
(see actual addresses)
Member
acquisition of 182-194,317-319
alias for 193
name 60
adding, changing, deleting, or replacing of 192-194
use of 317-319
obtaining address of 190-192,193
Messages
error 249,255,275,276,292,294
telecommunication 242
prefix of 242
type of 242
to computing system operator 111-113,314,316
to log 113
Metasymbols 20-21
Method, access 114,117,118
(also see BDAM, BISAM, BPAM, BSAM, QISAM, QSAM and QTAM)
MF operands
(see L- and E-forms)
Mnemonics
in conjunction with operation codes 316
in TESTRAN macro-instructions 245,252
value 16-18,303,308,309,313,315
Mode
load, in QISAM 194-204
buffer requirements 199
in connection with SYNAD routine 203
use of 194
locate-, in load-mode QISAM 195
locate-, in scan-mode QISAM 205,210,212
move-, in load-mode QISAM 195
move-, in QSAM 144
move-, in scan-mode QISAM 205,210
scan, in QISAM 204-213
Modules
load
in dynamically loaded programs 26,32,54,60-62,76-80
acquisition of 318-320,322
containing identified entry points 321
deletion of 62-63
linkage of 38
packing of 318
references to 318
in overlay programs 48
in programs being tested using TESTRAN 300,301
reusability attributes of 317
non-reusable 321
re-enterable 315,317,320-322
serially reusable 244,317,320-322
object
external names in 45
in programs being tested using TESTRAN 301

Nonreusable
(see modules)
NOTE macro-instruction 172-173
in relation to BSAM BSP macro-instruction 176
used with BSAM POINT macro-instruction 174

Object module (see module)
Only loadable (OL) 54
OPEN macro-instruction 122-124
operands permitted with BDAM 225
operands permitted with BPAM 182
operands permitted with BSAM 156,173,174
operands permitted with QSAM 133
purpose of 117
use of 155,171-172,194,203,212
use with SYSOUT writers 334
Opening
of job and link libraries 76
parallel 123
Operands 18,303-311
forms of 303
in L- and E-forms of macro-instructions 312-316
processing of 309-311
representation of 16
requirements for, ordinary and special 315-316
sublist 15,306,313
types of 14-15

keyword 14
positional 14,303,314
(also see format of macro-instruction)
Operation
asynchronous 91
input/output
(see GET, PUT, READ, and WRITE macro-instructions)
of programs 38
of tasks 39
of TESTRAN 244
parallel 34,317
Operator, computing system
communication with 93,105,111-113,314
Operator, relational 263
OPSW
(see program status word)
Option
alias 193
assembler 300
symbol table 300,301
buffer 123,227
control program 39-40,318-322
effect on macro-instructions 54,93,94
delete 195,210
error 141,143
exclusive 234
exit list 323
extended search 230,234
feedback 236
line spacing 333
linkage editor 301
stacker selection 161,333
TESTRAN editor 301
trailer label 327,332
Optional value 14
rules for 36
Output, system
(see writer)
Output, test 243,249,255
example of 294-299
formats of 275-294
Overflow
cylinder 195,197
independent area 195
printer carriage 155,176-177
record 210
track 137,160,176,187,226,231,236
Overlay
branch instructions 49,50
SEGLD macro-instruction 50
SEGWT macro-instruction 51
through CALL macro-instruction 48,49
when using TESTRAN 254,273,274
(also see path; segment)
Overlay programs 48-52
Overriding
data attributes 249
length 219,223

Pack
(see area)
Paper tape
(see tape)
PARAM operand 54,55,60,76,316
Parameter list
(see list)
Parameters 12-13
assembled 304,306
in data control block 118
loading responsibilities 306
packed 23-24
passing from control statements to program 36
passing from one program to another 28,35
problem program 55
supervisor 55
passing from problem program to supervisor 28
(also see list)
Parentheses 20
Partition 93
Partitioned data sets (see data sets)
Passing
control information 36
parameters 55
Path 49
Performance
optimization of 317
(also see execution)
Point, entry 26,28
identified 322
identified by IDENTIFY macro-instruction 54,63-64
identifier 34,45
name 45,318,319,321
referred to in ATTACH macro-instruction 75-84
referred to in CALL macro-instruction 41-44,49
referred to in DELETE macro-instruction 62-63
referred to in LINK macro-instruction 52-56
referred to in LOAD macro-instruction 60-62
referred to in XCTL macro-instruction 56-60
standard 318,321
Point, load
defined 178
POINT macro-instruction 174-175
in relation to BSAM BSP macro-instruction 176
used with BSAM NOTE macro-instruction 173
Pointers
in save areas 32,33,35
queue pointer field in queue control block (QCB) 95
Positional operands (see operands)
POST macro-instruction 91,93
Prefix 242
Printer
controlling 155,177-179,333
testing for carriage overflow 155,176-177
used for messages and comments 308
used for TESTRAN output 243,298
formats of printed output 276,294
Priorities
affect on WAITR macro-instruction 93
changing of 85
dispatching 80

limit 80
scheduling 39
Private library (see libraries)
Procedure, cataloged 300
Processing
of data set 114,117,141,194
end-of-volume 123
labels 123,327,332
sequentially 115,116,124,174,204
updating 212
of operands 18,309-310
of segments 38,49
of TESTRAN editor 243,268,294
stacked job 39
Processing unit, central
(see unit)
Program, source 243,244,247,249,255,260
Program interruption control area (PICA) 99
Program interruption element (PIE)
(see elements)
Program management
dynamic 52-64,317-322
overlay 48-52
simple 41-48
Program status word (PSW) (see word)
PRTOV macro-instruction 155,176-177
PUT macro-instruction
in load mode QISAM 194,195
locate mode 203-204
move mode 200-203
in QSAM 132,133,141,142,152
locate mode 145-146
move mode 146-147
substitute mode 147-148
in QTAM 239,242
PUTX macro-instruction
in QSAM 132,133,142
output mode 150-152
update mode 149-150
in scan mode QISAM 194,205
update mode 211-212,213

QISAM (queued indexed sequential access method) 194-213
load mode 194-204
scan mode 204-213
QSAM (queued sequential access method) 132-154,334
QTAM (queued telecommunication access method) 239-242
Queue, output 242
Queue control block (QCB)
described 95
used with ENQ and DEQ macro-instruction 94,96
Queue element (QE)
(see elements)
Quotation mark, single 18,36,307,308

R-type
(see macro-instructions)
READ macro-instruction
in BDAM 224,229-233
in connection with DCB
macro-instruction 227
in connection with WRITE
macro-instruction 234-236
in BISAM 213,218-221,222
in connection with WRITE
macro-instruction 223
in BPAM 182,187
in BSAM 155,164-165
in connection with BSP
macro-instruction 176
in connection with CHECK
macro-instruction 169-171
in connection with CNTRL
macro-instruction 178
in connection with DCB
macro-instruction 156,157,158,161
in connection with NOTE
macro-instruction 172-173
in connection with POINT
macro-instruction 174-175
L- and E-forms 316
Reading, backwards 164,171
Record, capacity 115
Record, logical
control characters in 155,333
deletion of
using QISAM 210,212
input of
using BISAM 218-221
using QISAM 209-210
using QSAM 141-145
length of 142,144,161,187,197,330
on direct-access devices 114-116
output of
using BISAM 222-223
using QISAM 200-204
using QSAM 145-148
processing of 117
updating of
using QISAM 211-212
using QSAM 149-152
Record, telecommunication 242
input of 241-242
output of 242
prefix of 242
Record, track descriptor 115
Record zero 114
Reenterable (see modules)
Registers
linkage 27
types of
parameter list register 27,28,35,76,312
parameter register 27,310
return code register 27,35,54
return register 26,27,28
save area register 27
supervisor parameter list register 27
location of parameters within 306
notation of 306-307
special 22,307,311,313
TESTRAN 252,307
responsibility for loading 306
Relative addresses
defined for direct-access devices
block identification 173
relative block 116
relative track 116
defined for magnetic tape
block identification 173

used with
BDAM 229,230,231,236
BPAM 189,191,193
BSAM 173,174
Release
of buffers 152-153,213,222,234
of key field 235
of main storage 71-74
of modules 60,62-63,321
Relexp 17
RELSE macro-instruction
152,153,194,205,213
Resource, serially reusable 94-95
Restoring
program interruption handling
procedures 99
registers 26
Restrictions 247,260,304,305,306
Result, test 263
Return codes
(see error analysis)
RETURN macro-instruction
26,33,34,35,38,45,46-48,54,91,143,169,254,
324-325
RETURN operand 260,267-268
Reusable
(see modules and resources)
Routine
standard system exit 100
synchronous exit 26,46,48
synchronous exceptional error exit
(SYNAD) 141,142,164,169,174,199,201,
208,209,212

S-type
(see macro-instructions)
Save area
(see area)
SAVE macro-instruction 26,31,38,44,45
in connection with RETURN
macro-instruction 46
Scale
(see attributes)
Scan-mode
(see QISAM
Scatter
(see loading of modules)
Scheduling
chained 142,144
of jobs 39
SEGLD macro-instruction 38,49,50-51
Segment
buffer 144,152,201,203
overlay 38,48-50,254
telecommunication 242
prefix of 242
SEGWT macro-instruction 38,49,51-52
Sense bytes 335
Sequence
calling 28,32,34,35,55
data set sequence number 329
test 243,244,260,263,265
volume sequence number 329
SET macro-instructions 245,246
SET COUNTER 269
SET FLAG 268-269
SET VARIABLE 269-270
SETL macro-instruction 194,207,208
use of 209,210,212
SF operands
(see L- and E-forms of
macro-instructions)
Special register notation
(see registers)
SPIE macro-instruction 36,97-101
STAE macro-instruction 38,48,101,102
Standard
page heading for TESTRAN edited output
(see heading)
status indicators
(see error analysis)
status information
(see information)
system exit routine
(see routine)
track format
(see format)
Status indicators
(see error analysis)
Status information
(see information)
Step, job
assignment of job step identification
331
conditional execution of 48
in connection with WAITR
macro-instruction 93
in dynamic program management
317,318,320,321
in testing programs 244,300,301
normal termination of 46,48
passing control information to 36
subpools within 99
timing of 39
use of modules within 60
STIMER macro-instruction 38,45,108-110
Storage, main
management 65-74
Store (ST) instruction 310,311
STOW macro-instruction 192-194
String, character 34
Sublist operands
(see operands)
Subpools
giving 81,82,83
releasing
FREEMAIN macro-instruction R-form 71
FREEMAIN macro-instruction S-form 72
requesting
GETMAIN macro-instruction R-form 65
GETMAIN macro-instruction S-form 67
sharing 81,82,83
subpool zero use 36,99
with ATTACH macro-instruction 75
Subprogram 99
Subset
of data management macro-instructions
155
of events being waited for 91
of value mnemonics 18
Subtask
creation 75-84
giving and sharing subpools with 80
limit and dispatching priorities of 80
passing parameters to 76

processing of 80
termination of 46,80,104
Supervisor
assistance in linkages
(see linkages)
services of 38
Supervisor call (SVC) instruction
in entry linkages 26
in macro-expansions 55,312
in return linkages 26
in TESTRAN programs
244,260,265,268,275,292
when identifiers are specified 34,54
Symbol 17
Symbol table 244,250,263,308
Synchronous
exit
defined 26
PRTOV 154,177
SYNAD
(see routine)
to a called program 34
operation
using CALL macro-instruction 41
using CHECK macro-instruction
169,171,174,177
using LINK macro-instruction 52
using RETURN macro-instruction 46
using SEGWT macro-instruction 51
using WAIT macro-instruction 89
using XCTL macro-instruction 56
SYSPRINT 301
System output (SYSOUT)
(see writer)
SYSUT1 301

Table
branch 28
of contents, volume 332
symbol 244,247,249,250,263,300,301,308
(also see task input/output table)
Tape, carriage control 154,176,177,178
Tape, magnetic
applicable operands for 123,133
block identification for 173
end of data set determination for 116
independence from 155
labels for (see labels)
mark 116,176,178
program control of 155,171,172,178
specification of 158
tracks on 158,327
use of in program testing 243
use of macro-instructions with
173,174,177
Tape, paper 158,161
Task
allocation of storage to 38
creation of 33,34,38,75
execution of 80,93,317
management of 75-88,320,321
passing parameters to 76
priorities of 80,93
queuing of 94-95
synchronization of 38,89-96
termination of 34,80,91,101-105
abnormally 54,102-105,124,141,169,210
normally 46-48,254
TESTRAN dumps of 249
use of modules within 54,60,62
Task control block (TCB)
completion code field 102
created 80
interrogated by EXTRACT
macro-instruction 87
Task input/output table (TIOT) address 87
Termination
(see task; step; and subtask)
Terminology 317
TEST macro-instructions 245,246,304,306
TEST AT 244,260-261
output lines for 292
TEST CLOSE 265-266,274
output lines for 292-293
TEST DEFINE 261
TEST ON 263-264
TEST OPEN 258-259,273-274
as affected by TEST AT
macro-instruction 260
as affected by TEST CLOSE
macro-instruction 292
effect of errors on 275
format of 245
maximum page count specification 301
output lines for 276,291
use of 244,260,265,300
TEST WHEN 262-263
TEST option
(see EXEC statement)
Testing 243
actions 243
procedure 243
(also see output; data; and registers)
Testing, input/output
for completion of input/output
operation 169
for printer carriage overflow 176,177
TESTRAN (test translator) 243-301
editor 18
interpreter 18
TEXT 18,245,308
TIME macro-instruction 38,107-108
Timer, interval 34,38,49
management 107-111
TLS 18,245
TRACE macro-instructions 202,245,246
TRACE CALL 255-256
output lines for 276,286-287
TRACE FLOW 253-255
output lines for 276,283-285
TRACE REFER 256-257
output lines for 276,287-290
TRACE STOP 257-258
output lines for 276,290-291
Tracing programs 243,254
(also see TRACE macro-instructions)
Track
capacity of 115
capacity record 115
descriptor record 115
labels on 332
(also see address; format; labels;
overflow; and tape)
TRUNC macro-instruction 152-153
TTIMER macro-instruction 38,111
TTRN 252

Type
(see attributes; and message)
Type I, II, III, and IV
(see linkages)
TYPE=T
(see CLOSE macro-instruction)

Underlined operands 20
Unit
central processing 80
check 335
control 115,223
exception 335
record device 169
(also see printer)
Unlabeled 174
Unmovable 115,156
Upper-case letters 17
USING statement 124,304,305

Value 18,313
Value, coded 14-15,303
Value, optional 14-15,313
rules for 36
Value mnemonics
(see mnemonics)
Vertical stroke 20
VL operand 36,54,55,60,76
Volume
disposition 124
end of 116,117,171,331
in QSAM and BSAM processing 115,116
in relation to relative track address 116
positioning 123,155,171,172,174
sequence number 329
serial numbers 116
switching 116,169,330
table of contents 332
when filled 105
(also see labels)

Wait
condition 93,94,95,169,177,218
flag 91
multiple 39
WAIT macro-instruction
for BDAM 231
for BISAM 219
for task synchronization 89-92,93
WAITR macro-instruction 92-93
Word
channel status 143,171,335
old program status (OPSW) 100
program status (PSW) 99,100,252
WRITE macro-instruction
in BDAM 224,233-237
in BISAM 213,222-223
in BSAM 155,157,161,166-167
checking 169-170,176,178
for updating 167-168
to create a direct organization data set 179-182
Write to operator or log
(see WTL, WTO, and WTOR macro-instructions)
Writer, system output (SYSOUT) 333-334
WTL macro-instruction 113
WTO macro-instruction 111-112
WTOR macro-instruction 112-113,314-316

XCTL macro-instruction 56-60,316,319
in programs being tested 254
use of in dynamic program management 32-35,46,76,321-322

Zero, record 114

READER'S COMMENTS

Title: IBM System/360 Operating System
Control Program Services

Form: C28-6541-1

Is the material:	Yes	No
Easy to Read?	___	___
Well organized?	___	___
Complete?	___	___
Well illustrated?	___	___
Accurate?	___	___
Suitable for its intended audience?	___	___

How did you use this publication?
___ As an introduction to the subject ___ For additional knowledge
Other _____

fold

Please check the items that describe your position:

___ Customer personnel	___ Operator	___ Sales Representative
___ IBM personnel	___ Programmer	___ Systems Engineer
___ Manager	___ Customer Engineer	___ Trainee
___ Systems Analyst	___ Instructor	Other _____

Please check specific criticism(s), give page number(s), and explain below:
___ Clarification on page(s)
___ Addition on page(s)
___ Deletion on page(s)
___ Error on page(s)

Explanation:

CUT ALONG LINE

fold

Name _____

Address _____

FOLD ON TWO LINES, STAPLE AND MAIL
No Postage Necessary if Mailed in U.S.A.

C28-6541-1

staple staple

fold fold

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
P.O. BOX 390
POUGHKEEPSIE, N. Y. 12602

ATTN: PROGRAMMING SYSTEMS PUBLICATIONS
DEPT. D58

fold fold

Printed in U.S.A. C28-6541-1

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York 10601

staple