File No. S360-36
Form C28-2004-2

TSS

IBM Systems Reference Library

IBM System/360 Time Sharing System

Assembler User Macro Instructions

IBM System/360 Time Sharing System provides comprehensive program and data management services which, together with communication, bulk output, and interruption handling services, are requested through macro instructions. These macro instructions are written in the assembler language as an aid to programming and processing time-shared tasks.

PREFACE

This publication contains a description of Time Sharing System/360 (TSS/360) macro instructions available to the assembler language user.

The publication is divided into three parts:

Part I: User Maco Instructions - contains an introduction to user macro instructions and their functional categories and describes the basic principals of the TSS/360 macro instruction language. Value mnemonics and basic macro instruction formats are discussed in detail.

Part II: Functional Macro Instruction Descriptions - contains detailed descriptions of the macro instructions available with TSS/360 within the framework of their major functional purpose.

Appendixes -- Use of exit routines, control characters available with certain data management facilities, and interrupt handling routines are explained.

All macro instructions available to the assembler language user are listed in this publication. However, since use of certain macro instructions requires detailed knowledge of system operation, these macro instructions are not of concern to the average TSS/360 user. Detailed descriptions are given in IBM System/360 Time Sharing System: System Programmer's Guide, Form C28-2008.

Prerequisite Publications

IBM System/360 Time Sharing System: Concepts and Facilities, Form C28-2003

IBM System/360 Time Sharing System: Assembler Language, Form C28-2000

Other recommended publications are:

IBM System/360 Time Sharing System: Linkage Editor, Form C28-2005

IBM System/360 Time Sharing System: Command System User's Guide, Form C28-2001

IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032

Third Edition (September 1968)

This editor has been updated technically; by including the Command System macro instructions, AETD, OBEY, PRMPT, MCAST, SYSIN, BPKD, and GDV; by adding the SIC operand to the GATE macro instruction descriptions; and by adding a return code to the list of return codes for the CDD macro instruction. In addition, it has been reorganized by placing macro instruction descriptions within the framework of their major functional categories. It should be noted that the DCB, OPEN, and CLOSE macro instructions which formerly appeared within each access method grouping now only appearance.

This edition is current with Version 3, Modification 0, and remains in effect for all subsequent versions or modifications of IBM System/360 Time Sharing System unless otherwise indicated. Significant changes or additions to this publication will be provided in new editions or Technical Newsletters. Before using this publication in connection with the operation of IBM systems, refer to the latest edition of IBM System/360 Time Sharing System: Addendum, Form C28-2043, for the editions of publications that are applicable and current.

Specifications contained herein are subject to change from time to time. Any such change will be reported in subsequent revisions or Technical Newsletters.

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.

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

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Time Sharing System/360 Programming Publications, Department 561, 2651 Strang Blvd., Yorktown Heights, N. Y. 10598

CONTENTS

PART I: USER MACRO INSTRUCTIONS 5
Section I: Introduction 5
Data Set Management 5
Program Management 6
Section II: The Macro Instruction Language 7
Macro Instruction Format 7
Name Field 7
Operation Field 7
Operand Field 7
Macro Description Value Mnemonics 9
Types of Macro Instructions 17
R-Type Macro Instructions 17
S-Type Macro Instructions 18
Other Macro Instructions 20

PART II: FUNCTIONAL MACRO INSTRUCTION DESCRIPTIONS 21
Section I: Data Set Management 22
Defining a Data Set to the System 22
DDEF -- Define a Data Set (S) 23
DCB -- Construct a Data Control Block (O) 25
CDD -- Retrieve and Execute DDEF Commands (S) 34
DCBD -- Provide Symbolic Names for a Data Control Block (O) 35
FINDDS* - Locate JFCB Corresponding to Data Set Name (S) 36
FINDJFCB* - Locate JFCB and Ensure Volume Mounting (S) 36
Connecting a Data Set To The System 37
OPEN -- Connect a Data Set to the System (S) 38
Accessing a Data Set 41
Virtual Sequential Access Method 42
GET -- Get a Record (R) 42
PUT -- Include a record in an Output Data Set (R) 43
PUTX -- Replace a Sequential Logical Record (R) 44
SETL -- Specify Start of Sequential Processing (R) 44
Virtual Indexed Sequential Access Method 46
GET -- Get a Record (R) 46
PUT -- Include a Record in an Output Data Set (R) 47
READ -- Read a Selected Logical Record (S) 48
WRITE -- Write a Selected Record (S) 49
SETL -- Specify Start of Sequential Processing (R) 51
ESETL -- Release Shared Data Set (R) 52
DELREC -- Delete a Record (R) 52
RELEX -- Release Read Exclusive Record (R) 53
Virtual Partitioned Access Method 55
FIND -- Find a Member of a Partitioned Data Set (S) 55
STOW -- Manipulate Partitioned Organization Directory (R) 57
Basic Sequential Access Method 61
READ -- Read a Block (S) 62
WRITE -- Write a Block (S) 64
CHECK -- Wait for and Test Completion of READ or WRITE Operation (R) 67
DQDECB -- Remove Unchecked DECBs From a Data Set's DECB Queue (R) 69
GETBUF -- Get a Buffer From a Pool (R) 70
FREEBUF -- Return a Buffer to a Pool (R) 71
GETPOOL -- Get a Buffer Pool (R) 72
FREEPOOL -- Free a Buffer Pool (R) 73
BSP -- Backspace a Block (R) 74
CNTRL -- Control On-Line Input/Output Devices (R) 75
FEOV -- Force End of Volume (R) 77
POINT -- Position to a Block (R) 78
NOTE -- Provide Position Feedback (R) 80
PRTOV -- Test for Printer Carriage Overflow (R) 81
Queued Sequential Access Method 83
GET -- Get a Logical Record (R) 84
PUT -- Include a Record in an Output Data Set (R) 85

PUTX -- Include a Logical Record in an Output or Updated Data Set (R) 86
RELSE -- Release an Input Buffer (R) 89
TRUNC -- Truncate an Output Buffer (R) 89
CNTRL -- Control a Printer or Stacker (R) 90
PRTOV -- Test for Printer Carriage Overflow (R) 91
SETL -- Specifies Start of Sequential Processing (R) 92
Input Output Request Facility 95
IOREQ -- Request an Input/Output Operation (S) 95
CHECK -- Wait for and Test Completion of an I/O Request (R) 98
VCCW -- Define a Virtual Channel Command Word (O) 98
Manipulating Entire Data Sets 101
Copying Data Sets 101
CDS -- Copy Existing Data Set (S) 101
Bulk Output Facilities 104
PR -- Print a Data Set (S) 104
PU -- Punch a Data Set (S) 107
WT -- Write a Data Set on Tape for Off-Line Printing (S) 109
Catalog Data Set Attributes 113
CAT -- Create or Change Catalog Entry (S) 113
DEL -- Delete Catalog Entry (S) 116
Disconnecting A Data Set From The System 118
CLOSE -- Disconnect Data Set From User's Problem Program (S) 118
VAM only 120
BSAM and QSAM only 120
Removing a Data Set From the System 123
ERASE -- Remove a Data Set from Direct-Access Storage (S) 123
REL -- Release Data Set or Remove Job Library From Program Library List (S) . . .124

SECTION II: PROGRAM MANAGEMENT 126
Virtual Storage Management 126
GETMAIN -- Allocate Virtual Storage (R) 126
FREEMAIN -- Release Allocated Virtual Storage (R) 128
CSTORE -- Control Section Store (S) 130
DCLASS* -- Specify Privilege Class (O) 131
RSPRV* -- Restore Privilege (O) 132
CKCLS* -- Check Protection Class (O) 132
LSCHP* -- List Changed Pages (R) 132
Program Loading and Linking 133
ADCON -- Generate an Adcon Group (O) 134
ADCOND -- Provide Symbolic Names for an Explicit Adcon Group (O) 137
ARM -- Initialize an Explicit Adcon Group (O) 138
CALL -- Call a Module (S) 139
LOAD -- Load and Retain a Module (R) 142
DELETE -- Delete a Loaded Module (R) 143
SAVE -- Save Register Contents (O) 144
RETURN -- Return to a Program (O) 146
DELET* -- Enter DELETE Service Routine (O) 148
DLINK* -- Dynamic Linkage Request (O) 148
ENTER* -- Enter a Privileged Routine (O) 148
INVOKE* -- Transfer Control (O) 148
LIBESRCH* -- Locate Program Module in External Library (S) 148
RESUME* -- Restore Registers (O) 148
STORE* -- Store Register Contents (O) 148
Interrupt Handling Facilities 149
SIR -- Specify Interrupt Routine (S) 150
SPEC -- Specify Program Entry Conditions (S) 152
SEEC -- Specify External Entry Conditions (S) 155
SSEC -- Specify Supervisor Call Entry Conditions (S) 157
SAEC -- Specify Asynchronous Entry Conditions (S) 158
STEC -- Specify Timer Entry Conditions (S) 161
SIEC -- Specify Input/Output Entry Conditions (S) 164
DIR -- Delete Interrupt Routine (S) 165
SAI -- Save and Inhibit (O) 166
RAE -- Restore and Enable (O) 167
INTINQ -- Interrupt Inquiry (O) 167
USATT -- Give User Control of Attention Interrupts (O) 169
CLATT -- Give System Control of Attention Interrupts (O) 170
AETD -- Create an Attention Entry Table (O) 170

ATPOL* -- Poll For Pending Attention Interrupt (nonstandard) 172
ITI* -- Inhibit Task Interrupts (nonstandard) 172
PTI* -- Permit Task Interrupts (nonstandard) 172
PCSVC* -- Enter Program Checkout Subsystem (nonstandard) 172
Transfer To Command Mode From Program Mode 173
PAUSE -- Enter Command Mode (R) 173
COMMAND -- Enter Command Mode (R) 174
EXIT -- Normal Program End (R) 175
ABEND -- Abnormal Task End (R) 176
OBEY -- Execute a Command or Command Statement (O) 177
CLIC* -- Read Command From SYSIN (Conversational) (O) 178
CLIP* -- Read Command From SYSIN (O) 178
RTBN* -- Create Privileged Linkage Queue Entry (O) 178
Communication Between User Program and SYSIN/SYSOUT 179
GATRD -- Get Record from SYSIN (S) 179
GATWR -- Write Record on SYSOUT (S) 182
GTWAR -- Write Record on SYSOUT and Read Response from SYSIN (S) 183
GTWSR -- Write Record on SYSOUT and Read Record from Terminal SYSIN (S) 184
SYSIN -- Obtain a Message From SYSIN or the Source List (S) 185
PRMPT -- Prompt System to Display a Particular Message (S) 188
MSGWR -- Issue Message and Get Response (S) 191
MCAST -- Modify Character and Switch Table (O) 193
Communication With Operator and System Log 197
WTO -- Write to Operator (S) 197
WTOR -- Write to Operator with Reply (S) 198
WTL -- Write to Log (S) 199
Timing Maintenance 200
STIMER -- Set Interval Timer (O) 200
TTIMER -- Test Interval Timer (O) 203
EBCDTIME -- Convert System Time into EBCDIC Format (S) 204
REDTIM* -- Read Time (O) 207

Command Creation 208
BPKD -- Create a Builtin Procedure Key (O) 208
GDV -- Get Default Value (R) 211
System Oriented User Macro Instructions 213
AWAIT* -- Tests for Event Completion and Return Control (O) 213
TWAIT* -- Tests for Completion of Event(O) 213
VSEND* -- Inter-Task Communication(O) 213
VSENDR* -- Inter-Task Communication with Response(O) 213
XTRSYS* -- E xtract From System Table(O) 213
XTRCT* -- Extract TSI Field(R) 213
XTRXTS* -- Extract From TSI (O) 213

APPENDIX A: EXIT LIST (EXLST) 215
Characteristics of Exit Routines 216
Exit-List Example 217

APPENDIX B: SYNCHRONOUS ERROR EXIT ROUTINE (SYNAD) 218
Entry To SYNAD During BSAM or QSAM Operations 218
Entry to SYNAD During VISAM Operations 221

APPENDIX C: END OF DATA ADDRESS (EODAD) 223

APPENDIX D: CONTROL CHARACTERS 224
Machine Code 224
Extended USASI Code 224

APPENDIX E: LINKAGE CONVENTIONS 225
Proper Register Use 226
Reserving a Save Area 226
Reserving a Parameter Area 227
Implicit Linkage 227
Explicit Linkage 228
Explicit Deletion 228

APPENDIX F: DATA CONTROL BLOCK FIELDS 229
Sources for Providing Data Set Attributes 229

Priority of Sources 230

APPENDIX G: DETAILED DESCRIPTION OF DDEF MACRO INSTRUCTION 237

APPENDIX H: MACRO INSTRUCTION GENERATION OF LITERALS 247

APPENDIX I: INTERRUPTION HANDLING FACILITIES 253
Establishing Interruption Routines 253
Processing an Interruption 255
Communication Area 255
Entry 255

APPENDIX J: THE TSS/360 SYSTEM MACRO AND COPY LIBRARY 256
System Macro and COPY Library Service Facilities 257
Generating the Library 257
Using Symbolic Libraries 257
Requesting Symbolic Library Services 258

APPENDIX K: SHARING VIRTUAL STORAGE DATA SETS 260
Types of Interlocks 260
Levels of Interlocks 260
User Considerations 261

APPENDIX L: OPEN/CLOSE GENERATED PARAMETER LIST 262

INDEX 263

FIGURES

Figure 1. Time Sharing System/360 Services 21
Figure 2. Save Area Layout and Word Contents 227
Figure 3. TSS/360 Interruption Handling Facilities 254
Figure 4. Information Available Upon Entry to an Interrupt Routine 256
Figure 5. System Macro and COPY Library Symbolic Component Format 257
Figure 6. Format of a Line in a Line Data Set 258

TABLES

Table 1. Value Mnemonics and Their Permissible Operand Forms 11
Table 2. Acceptable record formats for QSAM and the PUTX Macro Instruction . . 88
Table 3. Final Magnetic-Tape Positions 120
Table 4. Factors Determining Magnetic-Tape Positioning For BSAM and QSAM . . .121
Table 5. Return Codes from All GATE Macro Instructions 181
Table 6. Conditions Upon Exit -- Routine Entries 216
Table 7. Data Event Control Block (DECB) 220
Table 8. DCB Operands, Their Specification, Access Methods, and Alternate Sources (Part 1 of 2) 231
Table 9. Operands for DDEF Macro Instruction 238
Table 10. Literals Generated by Macro Instructions (Part 1 of 6) 249
Table 11. Effect of OPEN Options on Data Set Interlocks 262

SECTION I: INTRODUCTION

The TSS/360 user macro instructions provide two basic services; data set management and program management. These two services and the various management functions performed by each are summarized below.

DATA SET MANAGEMENT

- Define a Data Set to the System - by introducing a data set to a task and describing the characteristics or attributes of a data set, such as its record organization, disposition (i.e., OLD or NEW, etc.), and data set name, for future system use. TSS will subsequently (after the data set has been connected to the system) reference the indicated attributes to determine the appropriate access method routines and other control information.

- Connect a Data Set to the System - by making the attribute specifications, describing a data set, available to the system, thereby logically connecting the data set to the system. Appropriate access method routines are initialized, labels are processed (if specified), and the data set is positioned for user processing.

- Access a Data Set - by using the macro instructions associated with the appropriate VAM or SAM access method or provide your own input/output device management routines through use of the IOREQ macro facilities. A user can store, retrieve, or modify data sets using the macro instructions associated with the access method he uses.

- Manipulate an Entire Data Set - rather than individual records within a data set. An entire data set can be manipulated and transferred from one area of virtual storage to another, to punched cards, printer listings, or magnetic tape devices.

- Catalog Data Set Attributes - by recording certain predefined data set attributes in catalog entries so that the data set can be subsequently located by using only its name, without redefining all of its attributes to the system.

- Disconnecting a Data Set From the System - tells the system a user has finished processing a data set and, permanently or temporarily, disconnects the system from the control block (DCB) containing the description of the data set's attributes and access method specifications.

- Removing a Data Set From the System - causes a data set to be physically removed from the system and releases the storage areas, on which it was recorded, to the system for future use.

PROGRAM MANAGEMENT

- Virtual Storage Management - allows a user to acquire or release virtual storage in units of pages or 8 byte multiples, or to transform contiguous virtual storage bytes into an object module consisting of a single control section.

- Loading and Linking - macro instructions allow a user to explicitly or implicitly load program modules and establish standard linkage between calling and called program modules.

- Interrupt Handling Facilities - allow programmers to assume control at specific types of interrupts and execute special user coded interrupt servicing routines instead of the system provided interrupt servicing routines.

- Transfer to Command Mode - from program mode allows a user to interrupt a program's execution, either temporarily or permanently, and pass control to command mode for subsequent processing.

- Control Communication With SYSIN and SYSOUT - permits a user to pass data, messages, and commands, to and from a coded program to SYSOUT and SYSIN devices.

- Communication With Operator and System Log - allows a user to pass messages, issued during a program's execution, to the system operator, and to record those messages in the system operator's log.

- Timing Maintenance - provides a user with the ability to set timers which can measure the time of a task's execution or the elapsed calendar time.

- Command Creation - allows a user to create his own commands and, once created, issue them at his terminal.

- System Oriented User Macro Instructions - are available to all users, but are meant to be used only by system programmers; therefore, these macro instructions are only briefly mentioned here, but their detailed descriptions appear in the System Programmer's Guide.

SECTION II: THE MACRO INSTRUCTION LANGUAGE

Macro instructions for TSS/360 are processed by the assembler using IBM-supplied macro definitions.

Processing a macro instruction by the assembler is called the expansion of the macro instruction. Expansion results in fields of data and executable instructions, called the macro expansion. Fields of data, called parameters, specify the exact nature of the service to be performed and are contained in either registers (parameter registers) or data areas (parameter lists). If the parameters are contained in registers, only registers 0 and 1 may be used. If the parameters are contained in a parameter list, the address of that list is placed in register 1 and referred to by the called service routine.

MACRO INSTRUCTION FORMAT

System macro instructions, like assembler instructions, are written in this format:

Name	Operation	Operand

Name Field

The name field of the macro instruction may contain a symbol or remain blank. Normally, this symbol is the name associated with the first executable instruction of the macro expansion.

Operation Field

The operation field contains the mnemonic operation code of the macro instruction. This code may be a string of not more than eight alphameric characters, the first of which is alphabetic.

Operand Field

The operand field may contain no operands, or one or more operands separated by commas; the two types of operands are: positional and keyword.

POSITIONAL OPERANDS: Positional operands must be written in a specific order; for instance:

```
EXAMPLE   A,B,C
```

Assembly-time processing of operands A, B, and C is determined by whether they are the first, second and third operands, respectively. If

the second operand (B) is omitted, the user must supply the second comma to maintain the proper position for the third operand (C). Blanks may not be embedded in the positional operand field:

```
EXAMPLE   A,,C
```

If the last positional operands are omitted, delimiting commas need not be written. For example, if operands B and C are omitted, the macro instruction may be written:

```
EXAMPLE   A
```

KEYWORD OPERANDS: The keyword associated with a specific keyword operand uniquely identifies that operand to the assembler. Therefore, these operands may be written in any sequence. A keyword operand is written as a keyword, shown in each macro instruction description, immediately followed by an equal sign and its value:

```
EXAMPLE   AREA=X,LENGTH=100
```

MIXED OPERANDS: An operand field may contain both positional and keyword operands; however, all positional operands must precede all keyword operands. For example:

```
EXAMPLE   A,B,C,AREA=X,LENGTH=100
```

THE RULES FOR OMITTING POSITIONAL AND KEYWORD OPERANDS APPLY TO MIXED OPERAND FIELDS; IF OPERANDS B, C, AND AREA ARE OMITTED:

```
EXAMPLE   A,LENGTH=100
```

OPERAND SUBLISTS: A sublist is one or more positional operands, each separated by commas and the total list enclosed in parentheses. The entire sublist is considered as one operand in that it occupies a single position in the operand field or is associated with a single keyword. The contents of the sublist are processed similarly to positional operands.

The following operands are sublists:

```
(A,B,C)
(A)
```

Note that the sublist (A) above consists of only one operand. When a macro instruction description shows that an operand is written as a sublist, the enclosing parentheses must be written even if only one element appears in the sublist.

Macro Description Notational Symbols: Notational symbols in the operand field of macro instruction descriptions assist the user in showing how, when, and where an operand should be written. The notational symbols are: vertical stroke, shown as |; braces { }; brackets [ ]; ellipsis, shown as ..., and underscore _____.

1. Vertical stroke means "exclusive or." For example, A|B means that either the character A or the character B, but not both, may be written. Alternatives are also indicated by operands being aligned vertically, as shown in the next paragraph.

2. Braces denote grouping. They are used most often to group alternative operands or alternative operand forms. For instance, the following two operand descriptions are equivalent:

   {INPUT|OUTPUT}

   $\left\{\begin{matrix}\text{INPUT}\\ \text{OUTPUT}\end{matrix}\right\}$

3. Brackets denote options. Information enclosed in brackets may either be omitted or written in the macro instruction, depending on the service to be performed.

   In the following case, the operand of the EXAMPLE macro instruction is optional and need not be supplied. However, if the operand is supplied, it must be one of the alternatives grouped in braces.

Name	Operation	Operand
[symbol]	EXAMPLE	[mode-{INPUT|OUTPUT}]

4. An underscore means that if an operand is not specified, the underscored option is assumed. The underscored word, INPUT, in the above example indicates that INPUT is assumed if the operand is omitted.

5. The ellipis denotes the optional occurrence of the preceding syntactical unit one or more times in succession. If the syntactical unit consists of one term, it is followed by a comma and an ellipsis. For example,

   dcb-adr,...

   indicates that the term dcb-addr can be repeated with commas separating each term. No comma is placed after the last term.

   If the syntactical unit consists of more than one term, it is enclosed in braces -- {} -- to indicate the unit that may be repeated. The comma and ellipsis are placed outside the braces. For example,

   {dcb-addr,opt-code},...

   indicates that the unit dcb-addr,opt-code can be repeated with commas separating each unit. No comma is placed after the last unit.

6. Upper-case (capital) letters indicate the portions of the operand that must be written exactly as shown. For example, the operation field and coded values in the operand field must always be transcribed in upper-case letters.

7. Commas and parentheses must be written as shown in an operand field. They are delimiters, not notational symbols.

Macro Description Value Mnemonics

Value mnemonics help the user remember the forms a particular operand may assume. Eleven value mnemonics are used in this publication.

relexp
addr
addrx
addx
integer
absexp
value
text
code
symbol
characters
name
specsym
alphnum

In macro instruction descriptions in this publication, each positional operand is specified by a meaningful name hyphenated with a value mnemonic, as illustrated:

Name	Operation	Operand
[symbol]	EXAMPLE	name-value mnemonic

Each keyword operand is specified by the keyword, an equal sign, and a value mnemonic, as illustrated:

Name	Operation	Operand
[symbol]	EXAMPLE	KEYWI=value mnemonic

One or more operand forms may be substituted for each value mnemonic. For example, the value mnemonic, relexp, denotes that a relocatable expression may be written as the operand form; the value mnemonic, addx, specifies that an explicit address or an implied address may be written.

The 10 operand forms are:

relocatable expression
register notation
explicit address
implied address
symbol
decimal integer
absolute expression
code
text
characters
data set name
special symbol
alphameric characters

Table 1 lists the value mnemonics and their permissible operand forms. In the subsequent text each operand form is fully described.

Table 1. Value Mnemonics and Their Permissible Operand Forms

Operand Forms	Value Mnemonics													
	relexp	absexp	addr	addrx	addx	integer	value	text	code	symbol	characters	name	alphnum	specsym
Relocatable Expression	X		X											
Register Notation			X	X			X							
Explicit Address				X	X									
Implied Address (Indexed)				X	X									
Symbol										X				
Decimal Integer						X								
Absolute Expression		X					X							
Code									X					
Text								X						
Characters											X			
Data Set Name												X		
Alphameric Characters													X	
Special Symbol														X

Note: An X indicates that the operand form may be written.

Relocatable Expression: The value of a relocatable expression would change by n if the program in which it appears is relocated n bytes from its originally assigned storage area. All relocatable expressions must have a positive value. A relocatable expression may be a relocatable term. A relocatable expression may contain relocatable terms -- alone or in combination with absolute terms -- under the following conditions:

1. There must be an odd number of relocatable terms.

2. All relocatable terms but one must be paired. Pairing is described later in "Absolute Expression."

3. The unpaired term must not be directly preceded by a minus sign.

4. A relocatable term must not enter into a multiply or divide operation.

A relocatable expression reduces to a single relocatable value. This value is the value of the odd relocatable term adjusted by the values represented by the absolute terms and/or paired relocatable terms associated with it. The relocatability attribute is that of the odd relocatable term.

Complex relocatable expressions are also permitted. Refer to Assembler Language.

In the following examples of relocatable expressions, SAM, JOE, and FRANK are in the same control section and are relocatable; PT is absolute.

```
SAM
SAM-JOE+FRANK
JOE-PT*5
SAM+3
```

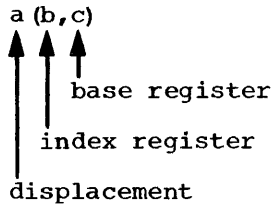
Note that SAM-JOE is not relocatable, because the difference between two relocatable addresses is constant.

Register Notation: Register notation is written as an absolute expression enclosed in parentheses. The absolute expression, when evaluated, must be some value 2 through 12, indicating the corresponding general purpose register.

In these examples of register notation, SAM and JOE are relocatable and PAL is absolute.

```
(5)         indicates register 5
(SAM-JOE)
(PAL)
(PAL+3)
```

Explicit Address: The explicit address is written in the same form as an assembler language operand:



Examples of explicit addresses are:

```
2 (0,5)
0 (2,4)
```

Implied Address (indexed): An implied address is written as a symbol, optionally indexed by a specified index register.

Examples of implied addresses are:

```
GUPOFF
ALPMAY (4)
```

Note that ALPMAY is indexed by register 4.

Symbol: A symbol may be a symbolic address (i.e., a single relocatable term), such as the name of an instruction in an assembler-language program, or it may merely be a character string used for identification, not location (such as the ddname parameter of a DCB macro instruction).

In TSS/360, the alphabetic characters are the letters A-Z, and $, @, and #. The alphameric characters are the alphabetic characters plus the digits 0-9.

The symbol is written as a string of up to eight alphameric characters, the first of which is alphabetic. Embedded commas and blanks are not permitted. Symbols beginning with the characters CHD may not be used, since symbols beginning with those characters are reserved for system use. Examples of symbols are:

```
DDNAME1
ROGER
```

```
LOOP12
START
#1
```

Decimal Integer: The operand may be written as a whole decimal number; e.g., 5, 31, 127, etc.

Absolute Expression: An absolute expression may be an absolute term or any arithmetic combination of absolute terms. An absolute term may be an absolute symbol or any self-defining term. All arithmetic operations are permitted between absolute terms.

An absolute expression may contain relocatable terms alone or in combination with absolute terms, under these conditions:

1. There must be an even number of relocatable terms in the expression.
2. The relocatable terms must be paired. Each pair of terms must have the same relocatability attribute; i.e., they appear in the same control section of an assembly. Each pair must consist of terms with opposite signs. The paired terms do not have to be contiguous, e.g., RT+AT-RT, where RT is relocatable and AT is absolute.
3. A relocatable term must not enter into a multiply or divide operation.

Pairing of relocatable terms (with opposite signs and the same relocatability attribute) cancels the effect of relocation. The value represented by the paired terms remains constant, regardless of program relocation.

Example: In the absolute expression A-Y+X, the term A is absolute, and the terms X and Y are relocatable with the same relocatability attribute. If A equals 50, Y equals 25, and X equals 10, the value of the expression becomes 35. If X and Y are relocated by a factor of 100, their values become 125 and 110. However, the expression still evalutes as 35 (50-125+110=35).

An absolute expression reduces to a single absolute value.

In these examples of absolute expressions, JOE and SAM are relocatable and defined in the same control section; BERNY and DAVE are absolute:

```
331
DAVE
BERNY+DAVE-83
JOE-SAM
DAVE*4+BERNY
```

Code: A code is written exactly as indicated in the macro instruction description. For example:

Name	Operation	Operand
[symbol]	FTBAL	scores-code

scores
specifies the desired action

```
TD - Touchdown
FG - Field goal
HT - Half-time is called
```

The macro instruction might be written in a program:

```
SAM     FTBAL    TD
        FTBAL    FG
DUME    FTBAL    HT
```

Text: A text operand is written as a string of characters enclosed in apostrophes. Embedded blanks and special characters are permitted. Two apostrophes or two ampersands must be used to represent one apostrophe or one ampersand in the character string. The text operand may not exceed 255 characters including the enclosing apostrophes. For example:

```
'AREA,PCB,132, ,1256'

'DO && DON''T'
```

Characters: The character operand is written as a character string. Embedded commas or blanks are not permitted. Two apostrophes or two ampersands must be used to represent one apostrophe or one ampersand in the character string. The character string may not be enclosed in apostrophes. For example:

```
CUBTDAVE+HEINZ+JOHN*830PMOT

DO&&DON''T
```

Data Set Name: The name of a data set or a group of data sets. The rules for writing data set names are presented below; the types of names that can be written for each macro instruction are described under each macro instruction's description.

Fully qualified name uniquely identifies one data set.

1. Stand-alone data set name identifies a data set that is not a member of a partitioned data set nor a generation of a generation data group. The name of a stand-alone data set is written as a series of symbols separated by periods. For example:

   ```
   DATASET.TRIAL.TEST1
   TERI.ROGER.LAURIE
   A.B.C.
   ```

   The rightmost symbol is the data set's simple name (TEST1, LAURIE, and C above); the other symbols are qualifiers. In TSS/360, for cataloging purposes, the maximum number of characters in a data set including periods, is 35. The maximum number of qualifiers for a one-character name is 17.

   Note: Data set names created under the IBM System/360 Operating System can contain a maximum of 44 characters; if data sets with names greater than 35 characters are to be cataloged in TSS/360, the user should employ the renaming facility of the CAT macro instruction or CATALOG command to define a suitable TSS/360 name.

2. Partitioned Data Set and Member Name identifies a data set that combines individual data sets, called members, into a single data set. The partitioned organization allows the user to refer to either the entire data set or to an individual member of the partitioned data set.

- The rules for writing the name of a partitioned data set are the same as for writing those of a stand-alone data set.

- The rules for writing a member name vary with each macro instruction that can manipulate members. Sometimes (as in LOAD and DELETE) only the simple member name (a symbol) is written. The full name is not required because the user has indirectly defined the partitioned data set (library) in which the module resides by assuring that the library is on the program library list prior to issuing those commands.

  The user could write

  ```
  LOAD SORTR
  ```

  if he has previously arranged that SORTR was in a library currently on the program library list.

  In other macro instructions (e.g., CDS), the user must give the fully qualified member name. This consists of the name of the partitioned data set suffixed by the simple member name in parenthesis. For example:

  ```
  HQW(ONETRY)
  G.H.AB(H)
  ```

  Here HQW and G.H.AB are partitioned data sets with members ONETRY and H, respectively.

  The name of the partitioned data set is written with the same rules as for a stand-alone data set. The parentheses and member name are merely considered as an appendage to that name.

3. Generation Names identify data sets which are part of a generation data group. These data sets can be referred to on an absolute or relative basis:

   a. Absolute Generation Names are written as the name of the generation data group followed by a period and the characters GxxxxVyy, where xxxx is a four-digit decimal generation number, and yy is a two-digit decimal version number. For example:

   ```
   HURST.LINER.TT.G0001V00
   HJ.LA4.WW.G0003V01
   HARQ.G0147V03
   ```

   The characters GxxxxVyy are considered a fixed-part of the overall name. The name of the generation data group is a partially qualified name applicable to all generations in the group.

   If the generation is a partitioned data set, a member (e.g., JOE) within that data set is referred to as follows:

   ```
   A.B.C.GxxxxVyy(JOE)
   ```

   b. Relative Generation Names are written as the name of the generation data group followed by the appropriate relative generation number enclosed in parentheses, as

   ```
   G.D.G(0)
   ```

   The relative generation number of the most recent generation is (0); the generation just prior to that is (-1); the one

before that is (-2), etc.; and a new generation to be added is (+1). For example:

```
GOST.UU.L19P(+1)
GOST.UU.L19P(-3)
MRQ.T.L5.SWIM(0)
```

If the generation is a partitioned data set, a member within that data set is referred to as follows:

```
SEAT(-3)(JOE)
```

where JOE is the member in question.

Partially qualified names refer to all data sets having the partially qualified name as their common higher-order qualifier.

1. Generation Data Group Name is the name that is common to each generation in the group. Generation data group names are restricted to a maximum of 26 characters including periods.

2. Other Partially Qualified Name can also be used to refer to two or more data sets. For example, the partially qualified name GO.AB14 can be used to refer to both of the following data sets: GO.AB14.A and GO.AB14.B. If these were the only two of a user's data sets with the same higher-order qualifier, GO.AB14, and he wished to erase them both, he could do so merely by specifying GO.AB14 in the ERASE macro instruction.

Special Symbol: A special symbol operand may consist of any string of from one to six alphameric or special characters (except for the tab, blank, comma, backspace, equal sign, and right and left parentheses). For example:

```
FORMNO
&H*/K
```

Alphameric Characters: An alphameric-character operand is written as a string of alphameric characters, the first of which need not be alphabetic. For example:

```
A00764
10E0D4
```

The limit on the number of characters is given in the description of each macro instruction in which it is used.

OPLIST OPERANDS: In a number of macro instruction descriptions in this publication, the operand field is specified as:

Operand
oplist-{text / addr}

This format implies that a list of keyword and/or positional operands may be written as fields of a character string. Also, the character string itself (enclosed in apostrophes) or the address of the string may be written as the oplist operand, depending on whether the text or addr form of the operand is chosen.

If oplist is presented as a character string (i.e., text operand form) the macro expansion places it in the assembled program followed

byan end-of-message code, and loads a pointer to the string in register 1. If oplist is given as an address (i.e., addr operand form) the expansion places that address in register 1. In this case, the user must define the operands elsewhere in the program and provide an end-of-message code.

To refer to and manipulate oplist macro instruction operands in coding, the address option of the operand is used, permitting the operand character string to be set up as a series of adjacent fields, each with its own label.

The string must end with a hexadecimal 27, which serves as an end-of-message code. Any unused space in each of the adjacent fields in the string must be filled with blanks to the maximum size of that field. Unlike other operand forms, all commas in an oplist operand must be written even if parameters are defaulted. A typical operand string might be coded:

```
OPLIST    DC   C'first operand'
OPLIST1   DC   C',second operand'
OPLISTN   DC   C',n operand'
          DC   X'27'
```

TYPES OF MACRO INSTRUCTIONS

Most system macro instructions are of two basic types: R-type (register) or S-type (storage). In this publication, the letter (R) or (S) follows the name of each macro instruction description to indicate its type. Macro Instructions that are neither R- nor S-type, referred to as "other" macro instructions, are denoted by (O) in their descriptions.

Some macro instructions generate literals in their expansions. Consequently, the rules for literal pool coverage must be followed. Refer to Appendix H, and to "Terms and Expressions" in Section 2 of Assembler Language.

R-Type Macro Instructions

An R-type (register) macro instruction is used when all required parameters can be contained in the two parameter registers, 0 and 1. An R-type macro instruction does not generate a parameter list; the parameters are placed in the parameter registers by instructions in the macro expansion. Execution time may be saved if the user places the data in the parameter registers as the result of previous operations before executing an R-type macro instruction.

Address operands in R-type macro instructions are always classified as addrx or addx. This arrangement allows the user to employ indexing, although the addresses passed in R-type macro instructions must be properly covered; i.e., the base register used for the passed address must contain the proper value to ensure that the address refers to the desired location in virtual storage.

For example, assume there is an R-type macro instruction, RTYPE, which will contain an address "area" in register 1 and the "length" of that area in register 0. Its external macro description would be:

Name	Operation	Operand
[symbol]	RTYPE	area-{addrx / (1)},length-{value / (0)}

Special Register Notation: The user's problem program might be written so that one or both of the parameters already exist in the proper parameter register when the macro instruction is issued. In this case, (1) or (0) is written as the operand. The notation (1) and (0) is referred to as special register notation. Registers 1 and 0 cannot be used in a macro instruction unless special register notation is shown in the macro instruction description.

S-Type Macro Instructions

An S-type (storage) macro instruction is used when the number of parameters to be passed to the called routine cannot be contained in the two parameter registers. The parameters are placed in a parameter list whose address is passed to the called routine in register 1.

There are three forms of the S-type macro instruction:

1. The Standard form
2. The L-form (Parameter list only)
3. The E-form (Executable code only)

Note: All S-type macro instructions may be written in L- and E-forms unless otherwise stated in the individual descriptions.

THE S-TYPE STANDARD FORM: The S-type standard form macro instruction generates both the parameter list required by the called routine and the linkage to that routine. If the S-type macro instruction is coded in a module that has a PSECT, the parameter list is generated in the PSECT. In this case, the PSECT must be properly covered by a base register. If the module has no PSECT, the parameter list is generated in-line and coding is generated to branch around it. If an S-type macro instruction is coded in a PSECT, the parameter list is generated in-line and coding is generated to branch around it.

Address operands in S-type standard form macro instructions are always classified as addr. Hence, they may not be indexed, and the user's problem program is not responsible for providing cover registers.

As an example, assume an S-type macro instruction, STYPE, that expects the addresses of two storage areas, "input" and "output," and the "length" of those areas. Its external macro description might be:

Name	Operation	Operand
[symbol]	STYPE	input-addr,output-addr,length-value

THE S-TYPE L-FORM: The L-form macro instruction creates a parameter list. E-form macro instructions then link to the service routine and point to the parameter list that is generated by the L-form macro instruction.

The assembler recognizes an L-form macro instruction by the keyword operand MF=L in its operand field.

Because the L-form macro instruction generates only a parameter list, operand types that require executable code, such as register notation, are prohibited.

There is an implied difference in the kinds of operands required in the external macro description when using the various forms of the S-type macro instruction. Where the standard form indicates addr and value operands (i.e., register notation is allowed), it is implicitly understood that L-form macro instructions allow only relexp and absexp operands (i.e., register notation is not allowed).

The external description of the L-form STYPE macro instruction becomes, by implication,

Name	Operation	Operand
symbol	STYPE	[[input-relexp],[output-relexp],[length-absexp],] MF=L

Note that the name field is required in the L-form because it usually becomes the label of the generated parameter list and is referred to by the E-form.

All operands of an L-form macro instruction are usually optional. It is assumed that operands that are omitted in the L-form will be supplied in the E-form macro instruction.

The L-form macro instruction generates the parameter list at the place the macro instruction is encountered. Because the L-form expansions contain no executable instructions, they should be placed in the program so that they do not receive control; e.g., among the DSs or DCs. An L-form macro instruction should never be written in a read-only control section.

THE S-TYPE E-FORM: A parameter list created by an L-form macro instruction, or by any other means, may be referred to by an E-form macro instruction. The user can update a parameter list by supplying operands in the E-form macro instruction.

The assembler recognizes an E-form macro instruction by the presence of the keyword operand in its operand field:

$$\text{MF=(E, list-}\left\{\begin{matrix}\text{addrx}\\(1)\end{matrix}\right\})$$

List should specify the location of the parameter list to be used by the E-form macro instruction. If (1) is written, register 1 should be loaded with the address of the L-form parameter list before execution ofthe macro instruction. The symbol in the name field of an L-form macro instruction becomes the name of the parameter list.

Once again, there is an implied difference in kinds of operands required. When standard form requires addr and value operands, the E-form requires addrx and value operands. The E-form thus allows the user to index addresses; however, proper cover registers must be provided.

The external description of the E-form STYPE macro instruction becomes, by implication,

Name	Operation	Operand
[symbol]	STYPE	[[input-addrx] , [output-addrx] , [length-value] ,] MF=(E,list-{addrx / (1)})

All operands are individually optional. The position of positional operands supplied in the E-form macro instruction causes the generation of executable instructions that replace the corresponding parameters in the parameter list of the L-form macro instruction with their new values.

Other Macro Instructions

The system macro instructions that cannot be classified as either R-type or S-type are referred to as "other", denoted by (O) in the macro instruction descriptions.

For example, the SAVE macro instruction does not produce parameters that pass to a called program. Its expansion results in instructions in the user's program that completely perform the requested service. Similarly, the DCB macro instruction only defines a data area. It is, in effect, an implied S-type L-form macro instruction.

The major functional groups into which macro instructions fall are data set management and program management. A summary of these functional groupings is indicated in Figure 1

TIME SHARING SYSTEM/360
ASSEMBLER USER MACRO INSTRUCTIONS

DATA SET MANAGEMENT		
Defining Data Set(s)		
DDEF	CDD	FINDDS*
DCE	DCBD	FINDJFCB*
Connecting Data Set(s) to System		
OPEN		
Accessing Data Set(s)		
VSAM	VISAM	VPAM
GET	GET	FIND
PUT	PUT	STOW
PUTX	READ	
SETL	WRITE	IOREQ
	SETL	IOREQ
PSAM	ESETL	CHECK
READ	DELREC	VCCW
WRITE	RELEX	
CHECK		
GETPOOL	QSAM	
GETBUF	GET	
FREEBUF	PUT	
FREEPOOL	PUTX	
BSP	RELSE	
CNTRL	TRUNC	
FEOV	CNTRL	
POINT	PRTOV	
NOTE	SETL	
PRTOV		
Manipulating Entire Data Set(s)		
Copying Data Set(s)	Bulk Output	
	PR	
	PU	
CDS	WT	
Catalcging Data Set Attributes		
CAT		
DEL		
Disconnecting Data Set(s) From System		
CLOSE		
Removing Data Set(s) From System		
ERASE		
REL		

PROGRAM CONTROL MANAGEMENT		
Virtual Storage Management		
GETMAIN	DCLASS*	CHKCLS*
FREEMAIN	PSPRV*	LSCHP*
CSTORE		
Program Linking and Loading		
ALTCND	DELETE	ENTER*
ADCON	SAVE	RESUME*
ARM	RETURN	LIBESRCH*
CALL	DELET*	STORE*
LOAD	DLINK*	
Interrupt Handling		
SIR	SIEC	USATT
SPEC	DIR	AETD
SSEC	SAI	ATPOL*
SEEC	RAE	ITI*
SAEC	INTINQ	PTI*
STEC	CLATT	PCSVC*
Transfer to Command Mode		
PAUSE	ABEND	CLIC*
COMMAND	OBEY	CLIP*
EXIT	RTRN*	
Communication With SYSIN/SYSOUT		
GATRD	GTWSR	MSGWR
GATWR	SYSIN	MCAST
GTWAR	PRMPT	
Communication With Operator and Log		
WTO		
WTOR		
WTL		
Timing Maintenance		
STIMER	EPOCTIME	
TTIMER	REDTIM*	
Command Creation		
BPKD		
GDV		
System Oriented Macro Instructions		
AWAIT*	VSENDR*	XTRXTS*
TWAIT*	XTRSYS*	XTRCT*
VSEND*		

*Although these instructions are available to all users, they are employed primarily by system programmers; therefore, refer to System Programmer's Guide, Form C28-2008, for a detailed description of these macro instructions.

Figure 1. Time Sharing System/360 Services

SECTION I: DATA SET MANAGEMENT

This section describes TSS/360 macro instructions available to the user to facilitate data set management. To enhance user understanding of these macro instructions, they are presented in functional groups that reflect their primary use in the system.

DEFINING A DATA SET TO THE SYSTEM

Certain characteristics of a data set must be described to TSS/360 Data Set Management and Task Management routines before a user may employ those management facilities to process and manipulate his data sets. These data set attributes can be furnished to the system from two to six different sources depending on whether the data set is a new data set or a data set that has been previously defined to the system. The various sources and their priorities are described in detail in Appendix F. The two major sources (i.e., and the only mandatory sources) provided for users to facilitate describing these data sets to the system, are the DDEF and DCB macro instructions respectively. These macro instructions and the CDD and DCBD instructions, which can be used with them, are briefly described below.

DDEF describes certain attributes of a data set to the system and defines or introduces that data set to a single task. Every data set referenced within the framework of any one task (i.e., from LOGON to LOGOFF) must be defined to that task via system or user issuance of the DDEF macro instruction (or command). In addition to providing unique attribute information, such as DSNAME, which cannot be supplied by any of the other sources for attributes, the DDEF macro instruction or command can also be used to furnish any attributes which are not furnished by the DCB macro instruction.

DCB reserves a space in virtual storage in which the attributes of the data set to be processed are to be placed and optionally describes the attributes of the data set (in conjunction with the DDEF macro instruction or command) to the TSS/360 management facilities.

CDD retrieves one or more DDEF commands from a line data set (created by issuance of a DATA or MODIFY command) that consists of prestored DDEF commands only. The CDD macro instruction (or command) processes these commands as if they were just issued at the terminal and thereby defines their related data sets to the system in the same manner as the DDEF macro instruction.

DCBD used to facilitate easier processing or modification of the data control block created by a DCB macro instruction. The macro instruction generates a dummy control section that provides the user with the symbolic names used by the system for referencing the fields in a data control block. A user can then use these labels to address the fields of any data control block he connects to the Dummy control section via a USING instruction.

Detailed explanations of each of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to defining a data set and the related macro instructions can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

DDEF -- Define a Data Set (S)

The DDEF macro instruction defines a data set and describes its characteristics to the system. Every data set that is referred to by an object program during execution must be defined by a DDEF macro instruction or command. All public VAM data sets are automatically cataloged at DDEF time. The system creates the catalog entry and provides the user with unlimited access. Each DDEF macro instruction is valid only during the session in which it is issued; thus data sets defined for one session must be redefined at every session that involves reference to them.

Note: The following description applies to the DDEF macro instruction used to define a standard data set on a public volume. (The standard data set is one that is VAM organized, on direct-access public storage, arranged in units of pages, and has standard labels.) To define non-standard public data sets or any private data set, refer to the detailed description of the DDEF macro instruction given in Appendix G.

Name	Operation	Operand
[symbol]	DDEF	oplist-{text | addr}

oplist
specifies the list of operands. They are:

Oplist
{ddname-symbol | PCSOUT} [,dsorg-{VI | VS}] ,DSNAME=name [,DISP={OLD|NEW}]

ddname
specifies the symbolic data definition name associated with this data set definition. It provides the link between the data control block in the program and the data set definition. It must contain one to eight alphameric characters, the first of which must be alphabetic. The user is not allowed to use a ddname that begins with SYS, since system reserved ddnames are prefixed with those characters.

PCSOUT
specifies that the program checkout subsystem is being used and a data set is being defined for dumps. A PCSOUT type of DDEF command or macro instruction is required in a task if the DUMP command is to be employed.

dsorg
specifies the organization of the data set.

VI
specifies the data set organization as virtual index sequential.

VS
specifies the data set organization as virtual sequential.

Note: If neither VI nor VS is specified, the data set organization assigned at system generation time is assumed.

DSNAME
specifies the name of the data set being defined; i.e., the name under which the data set may be cataloged or temporarily referred to.

This operand can be specified as the fully qualified name of: a partitioned or nonpartitioned data set, a member of partitioned data set, or a partitioned or nonpartitioned generation of a generation data group (identified by an absolute generation name or relative generation number).

DISP
specifies the status of the data set. If DISP is defaulted in a DDEF for an existing cataloged public data set, the system will assume a value of OLD. If DISP is defaulted for any data set which does not yet exist, the system will assume a default value of NEW. It should be noted that for existing uncataloged private data sets the DISP value must be explicitly specified as OLD. If the user tries to default such a data set, a DISP value of NEW is assumed and a system error results. The various defaults and options are summarized below:

NEW - for a new data set.
OLD - for an old data set.
Defaults - OLD - for old cataloged data sets.
NEW - for a new data set or for an old uncataloged private data set.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: All three of the DDEF operands shown above -- except VI -- are required for a cataloged data set. Only the ddname and data set organization are needed for an uncataloged data set. In either case, the data set conforms to the current installation standards.

Before the user can employ the DUMP command in his task, he must issue a PCSOUT type of DDEF macro instruction or command. Such a DDEF macro instruction or command requires PCSOUT as the first operand, followed by the dsname operand. Since the dump data set will be new, the DISP operand is defaulted.

The DDEF macro instruction or command causes a system entry to be established for the DDEF information so that allocation routines and access methods can refer to it. The link between this information and the problem program's reference to the data set (i.e., the data control block) is the data definition name, ddname. The entry containing the DDEF information is maintained until the task is concluded or until, through the RELEASE macro instruction or command, the data set is released.

The DDEF macro instruction or command may be used in conversational and nonconversational tasks.

If the user's problem program is being executed in conversational mode and an undefined ddname is referenced, prompting messages for DDEF operands are issued to the user regardless of confirmation option.

The user may change the ddname assigned in a previous DDEF macro instruction or command by using a DDEF macro instruction with a new ddname. The only operands used in this case are ddname, dsname, and disposition (OLD). The new ddname is then assigned and the old ddname eliminated.

At completion of execution of the DDEF macro instruction, the low-order byte of register 15 contains one of the following codes:

Code (Hexadecimal)	Significance
00	No error
04	Data set name undefined
08	Data set name not unique
0C	Attention interruption
10	DSORG inconsistent
20	Space not available
40	ddname not unique
80	Other

L- AND E-FORM USE: The oplist operand is required in the L-form of this macro instruction and is not permitted in the E-form. Only the text form of the operand may be used in the L-form of the macro instruction.

DCB -- Construct a Data Control Block (O)

The DCB macro instruction is one of the major sources (see Appendix F) by which the attributes of a data set may be described to the system. The attributes of a data set which can be provided via this macro instruction and the formats in which particular attributes can be specified are indicated below by access method.

Format: The format of the DCB macro instruction varies depending on the data set organization and the access method which is to be used, or was previously used, to perform I/O on that data set. All of the possible parameters which might be specified by a nonprivileged user in a DCB macro instruction are indicated by applicable access method.

Name	Operation	Operands	Applicable Access Methods					
			VSAM	VISAM	VPAM	BSAM	QSAM	IOREQ
[symbol]	DCB	[,DDNAME=symbol]	X	X	X	X	X	X
		[,DSORG=code]	X	X	X	X	X	X
		[,RECFM=code]	X	X	X	X	X	X
		[,LRECL=absexp]	X	X	X	X	X	
		[,EODAD=symbol]	X	X	X	X	X	
		[,SYNAD=symbol]		X	X*	X	X	X
		[,PAD=absexp]		X	X*			
		[,RKP=absexp]		X	X*			
		[,DEVD=code]		1.	1.	X	X	X
		[,KEYLEN=absexp]		X	X*	X		X
		[,DEN=absexp]				X	X	X
		[,TRTCH=code]				X	X	X
		[,PRTSP=absexp]				X	X	X
		[,MODE=code]				X	X	X
		[,STACK=absexp]				X	X	X
		[,MACRF=code]				X	X	
		[,BLKSIZE=absexp]				X	X	
		[,OPTCD=code]				X	X	
		[,IMSK=code]				X	X	
		[,EXLST=symbol]				X	X	
		[,NCP=absexp]				X		X
		[,BUFNO=absexp]				X		
		[,BFALN=code]				X		
		[,BUFL=absexp]				X		
		[,BUFTEK=code]				X		
		[,BUFCB=addr]				X		
		[,EROPT=code]					X	

* = VISAM members of a partitioned data set
1. = a value is assumed by the system

DDNAME (all access methods)
specifies the symbolic data definition name associated with a particular data set. This symbol provides the link which connects the

attributes of the data set defined by the DCB macro instruction with those specified by the DDEF macro instruction (or command), thereby providing the system with all the attributes necessary for processing the data set.

Specified as: A symbolic name of one to eight alphameric characters, the first of which must be alphabetic. The name specified for this parameter must be identical to the DDNAME parameter of the DDEF macro instruction that defines this data set. The only alternate source for this information is the user's program.

DSORG (all access methods)
specifies the organization of the data set.

Specified as: The various data set organizations. The codes by which they can be specified, and the access methods with which they are applicable are indicated below.

Code	Organization	Applicable Access Methods
PS	-- a physical sequential organization	BSAM,QSAM
PSU	-- a physical sequential unmovable organization in which the data set contains location-dependent information with respect to this data set. Treated as PS by TSS/360.	BSAM,QSAM
VS	-- virtual sequential organization	VSAM
VI	-- virtual indexed sequential organization	VISAM
VP	-- virtual partitioned organization	VPAM
VIP	-- virtual partitioned index sequential member of a partitioned organization	VPAM
VSP	-- virtual partitioned sequential member of a partitioned organization	VPAM
RX	-- I/O request facility is being used	IOREQ

For an existing VP data set, only VP need be specified. The organization of the member (virtual sequential or virtual index sequential) is determined by FIND and placed in the DCB. However, when creating a new member, the user must specify either VIP or VSP.

This information can also be supplied by the user's program or the DDEF macro instruction (or command), but must be supplied before issuing an OPEN macro instruction.

RECFM (all access methods)
specifies the format of the records in the data set.

Specified as:

For BSAM and QSAM:
U [T] [A|M]
V [B|T] [A|M]
F [B|S|T|BS|BT|BST|ST] [A|M]

Where the record format is:

U -- undefined-format records
V -- variable-length records
F -- fixed-length records

Where the physical attributes are:

B -- blocked records
S -- standard data set; no truncated blocks or unfilled tracks
T -- track overflow employed

Where the record contains:

A -- USASI control character
M -- machine code control character

Refer to Appendix D for a discussion of control characters.

Absence of any of the physical attribute mnemonics implies the opposite of that attribute. For instance, writing RECFM=V implies: variable-length, unblocked records, no control character, and no track overflow feature.

This information can also be supplied by the user's program, the DDEF macro instruction (or command), or the data set label.

For VAM data sets: All VAM data sets can be organized as fixed or variable length records but only VSAM and VPAM records can be specified as having undefined formats.

U[A|M] (applicable to VSAM, VPAM only)
V[A|M] (applicable to VSAM, VISAM, or VPAM)
F[A|M] (applicable to VSAM, VISAM, or VPAM)

Where the record format is:

U -- undefined-format records
V -- variable-length records
F -- fixed-length records

Where the record contains:

A -- USASI control character
M -- machine code control character

If A or M is not specified, no control character is assumed. Refer to Appendix D for a discussion of control characters.

For IOREQ

U -- undefined-format records

This information can also be supplied by the user's program, the DDEF macro instruction (or command), or the data set label.

LRECL (VAM, BSAM, and QSAM)
specifies the length in bytes of a logical record. For format-F records, this operand specifies the length of each record in the data set. For format-V and -U records, the user must insert the maximum expected value before the data set is opened. The maximum size is 32,766 bytes for BSAM, 1,048,576 bytes for VSAM, and 4000 bytes for VISAM. When reading format-U or -V records, the corresponding field in the data control block (DCBLRE) contains the length in bytes of the record just read.

This information can also be supplied by the user's program, the DDEF macro instruction (or command), or the data set label.

EODAD (VAM, BSAM, and QSAM)
specifies the address of the user's end-of-data routine for input data sets. This routine is entered if the user requests a record when there are no more records in the data set. If no routine has been provided, and the end-of-data condition has been encountered, the task is abnormally terminated. (Refer to Appendix C.)

If the symbol supplied is an external symbol, it must also appear as the operand of an assembler language EXTRN statement in the same program module as the DCB macro instruction.

The only alternate source for this information is the user's program.

SYNAD (VISAM, VISAM members, BSAM, QSAM, or IOREQ)
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 the system encounters a condition that would cause control to be given to the SYNAD routine, the task is abnormally terminated.

The only alternate source for this information is the user's program.

If the address specified is an external symbol, the symbol must also appear as the operand of an assembler language EXTRN statement in the same program module as the DCB macro instruction.

PAD (VISAM or VISAM members)
specifies the percentage of space (to a limit of 50 percent) to be left available within the pages of a virtual index sequential data set, thus providing for insertions within the pages.

This information can also be supplied by the user's program, the DDEF macro instruction (or command), or the data set label.

RKP (VISAM or VISAM members)
specifies the displacement (relative key position) of the key field from the first byte of a logical record.
Note: For format-V records, the logical record includes the length field as the first four bytes.

This information can also be supplied by the user's program or the DDEF macro instruction (or command).

DEVD (BSAM, QSAM, IOREQ, or VAM)
specifies the device on which the data set resides. Additional keyword operands are available, as shown below, to provide device-dependent information to device-dependent parameter bytes in the data control block.

Note: For VAM, DA is assumed, and the user can supply the KEYLEN operand if desired.

```
  DA  [,KEYLEN=absexp]
  TA  [,DEN={0|1|2}]  [,TRTCH={C|E|T|ET} ]
  PR  [,PRTSP={0|1|2|3}]
{RD}
{  }   [,MODE= {C|E}]  [,STACK= {1|2}]
{PC}
```

Note: Since nonprivileged users cannot address unit record devices directly, they may not specify PR (printer), RD (card reader), or PC (card punch). These devices may be specified only by users with proper system authorization.

This information can also be supplied by the user's program, the DDEF macro instruction or command, or the data set label.

DA

specifies a direct-access device.

KEYLEN (VISAM or VISAM members, BSAM, or IOREQ)
specifies length in bytes of the key associated with a physical record. When a record is read or written, the number of bytes transmitted is equal to key length plus record length. Maximum value of the key is 255.

This information can also be supplied by the user's program or the DDEF macro instruction or command.

TA

specifies magnetic tape.

DEN (BSAM, QSAM, or IOREQ)
specifies a value for the tape recording density in bits per inch as listed below.

DEN Value	Tape Recording Density (bits/inch)	
	Model 2400 Tape Drive	
	7-Track	9-Track
0	200	--
1	556	--
2	800	800

This information can also be supplied by the user's program or the DDEF macro instruction (or command).

TRTCH (BSAM, QSAM, IOREQ)
specifies, for 7-track tape, recording technique, where:

C -- Data conversion feature available. If data conversion is not available, only format-F and format-U are supported.
E -- Even parity is used.
T -- BCD to EBCDIC translation is required.

This information can also be supplied by the user's program or the DDEF macro instruction (or command). If not supplied by any source, odd parity and no translation is assumed.

PR

specifies printer.

PRTSP (BSAM, QSAM, or IOREQ)

specifies the line spacing on a printer. Either 0, 1, 2, or 3 may be specified.

0 = No spacing
1 = Space one line
2 = Space two lines
3 = Space three lines

This information can also be supplied by the user's program or the DDEF macro instruction or command. If not supplied by any source, 1 is assumed.

RD
specifies card reader.

PC
specifies card punch.

MODE (BSAM, QSAM, or IOREQ)
specifies the mode of operation for a card reader or a card punch, as follows:

C - the card image (column binary) mode
E - The EBCDIC code

This information can also be supplied by the user's program or the DDEF macro instruction (or command).

MACRF (BSAM and QSAM only)
specifies the type of macro instructions to be used in processing a particular data set.

Specified as:

For BSAM:

(R[C|P]) | (W[C|P]) | (R[C|P],W[C|P])

R -- READ macro instructions
W -- WRITE macro instructions

Optional modifiers:

C -- CNTRL macro instruction
P -- POINT macro instruction

For QSAM:

(G[S|C|SC]) | (P[S|C|SC]) | (G[S],P[S])

G -- GET macro instructions
P -- PUT macro instructions

Optional modifiers:

S -- SETL macro instruction
C -- CNTRL macro instruction

This information can also be supplied by the user's program or the DDEF macro instruction (or command).

BLKSIZE (BSAM only)
specifies a decimal value for the maximum block length in bytes. Maximum value of BLKSIZE is 32,760.

This information can also be supplied by the user's program, the DDEF macro instruction (or command), or the data set label.

OPTCD (BSAM or QSAM)
specifies an optional service to be provided. This service consists of performing a write validity check (for direct-access device only).

This information can also be supplied by the user's program, the DDEF macro instruction (or command), or the data set label. If not supplied by any source, the service is not performed.

IMSK (BSAM or QSAM)
specifies a four-byte hexadecimal number whose bit pattern indicates which system error handling procedures, if any, are to be invoked.

If FFFFFFFF is written, the system applies all optional error recovery procedures. This is the default condition.

If 00000000 is written, the system is to apply none of its optional error recovery procedures.

If any other four-byte hexadecimal number is written, the system applies its error recovery procedures only for those entries, set to 1 in IMSK, that correspond to error.

The first two bytes correspond to the first two bytes of the channel status word, and the other two bytes correspond to the first two sense bytes. Bit positions in each byte for specification of system error recovery procedures are:

XXXXXXAB XCXXXXXD YEFGHIYY YYYYYYYY

where a 1-bit in a given position indicates that the system is to handle the associated error condition:

X = System never tests this bit to determine entry to retry routines
Y = Device-dependent conditions
A = Unit check
B = Unit exception
C = Incorrect length

EXLST (BSAM or QSAM)
specifies the address of an exit list supplied by the user. See Appendix A for explanation of the exit list.

This information can also be supplied by the user's program.

NCP (BSAM)
specifies the number of consecutive READ or WRITE macro instructions issued before a CHECK macro instruction. This number may not exceed 99.

This information can also be supplied by the user's program or the DDEF macro instruction (or command).

BUFNO (BSAM)
specifies the number of buffers to be assigned to data control block. The number, expressed as a binary value, may not exceed 255.

This information can also be supplied by the user's program or the DDEF macro instruction (or command).

BFALN (BSAM)
specifies boundary alignment of buffers. This field is ignored in

Time Sharing System/360. Every buffer is automatically aligned on a doubleword bounndary.

This information can also be supplied by the user's program or the DDEF macro instruction or command.

BUFL (BSAM)
specifies a decimal number which is the length in bytes of each buffer to be obtained for a buffer pool. Maximum value is 32,760.

This information can also be supplied by the user's program or the DDEF macro instruction (or command). If not supplied by any source, the length is considered equal to the BLKSIZE operand.

BUFCB (BSAM)
specifies the address of a buffer control block.

This information can also be supplied by the user's program.

BFTEK (BSAM)
specifies that simple buffering is to be employed.

S - simple buffering

In simple buffering, a data set is associated with a specific group of buffers. A data set always uses buffers obtained from the pool assigned to its data control block at the time it is opened. Records can be moved between a buffer and an independent work area, processed within a buffer, or moved from an input buffer to an output buffer.

This information can also be supplied by the user's program or the DDEF macro instruction (or command). If not supplied by any source, BFTEK=S is assumed.

EROPT (QSAM)
When using GET/PUT macro instructions to process a sequential data set, an I/O error may occur. The user may specify one of three automatic error options to be used if there is no SYNAD routine or if the SYNAD routine returns control to the user's program. One of the following choices of action can be specified:

ACC -- accept the erroneous block and continue processing
SKP -- skip the erroneous block and process the next record
ABE -- abnormally terminate the task

Note: If the EROPT and SYNAD fields are not completed, the ABE option is assumed.

The choice of action that can be specified depends on which processing method (option) is specified in the OPEN macro instruction for the data set. The allowable combinations are as follows:

Action Operand	OPEN Option
ACC	INPUT,OUTPUT (for printer only), RDBACK, or UPDAT
SKP	INPUT,RDBACK, or UPDAT
ABE	INPUT,OUTPUT,RDBACK, or UPDAT

PROGRAMMING NOTES: During the assembly of a source program, the DCB macro instruction reserves storage space in a user program in which the attributes of a data set being described to the system may be subsequently placed. This storage area is known as a Data Control Block (DCB) and is created at assembly time, in line, wherever the DCB macro instruction appears in a user's source program. The reserved control block has a fixed length and consists of two contiguous parts: a common

portion, in which all information that is access method independent is to be placed, and an access method dependent portion.

In addition to furnishing the storage area for holding the attribute data describing a data set, the DCB macro instruction can also be used optionally, at execution time, to actually specify many of a data sets attributes. A user might furnish the system with such data attribute information as, the data set organization, its record format, whether or not buffering is to be used during I/O operations, the type of device the data set resides on, and the addresses of user written routines for handling I/O errors, processing labels, end-of-data-set processing, and Data Control Block modification routines. Any such attributes, specified with a DCB macro instruction are automatically placed in appropriate positions in the reserved storage area.

When the storage area reserved by the DCB macro instruction is filled with the attributes of a data set, it becomes the principal control block used to supply the system with information describing a particular data set or device. Once optionally specified attributes have been placed in the control block, the DCB routine returns to the user's program. All data management macro instructions, provided with TSS/360, reference this control block for pertinent data when they are executed.

CDD -- Retrieve and Execute DDEF Commands (S)

The CDD macro instruction retrieves one or more DDEF commands from a line data set containing prestored DDEF commands. The macro instruction processes the retrieved commands as if they had just been entered by the user. The user can thus create a line data set of commonly used DDEF commands with reference through the CDD macro instruction to eliminate direct DDEF macro instruction or command entries for each run of a program.

Name	Operation	Operand
[symbol]	CDD	oplist-{text / addr}

oplist
specifies the list of operands. They are:

Oplist
dsname-name[,DDNAME=ddname-symbol,...]

dsname
specifies the name of the line data set containing the prestored DDEF commands.

This operand can be specified as the fully qualified name of: a nonpartitioned data set, or a nonpartitioned generation of a generation data group (identified by absolute generation name or relative generation number).

DDNAME
specifies the following symbol as ddname of a particular DDEF command in the data set.

ddname
specifies the ddname of a particular DDEF command in the data set.

CAUTION: The user must make sure that none of the DDEF commands or macro instructions for his task has the same ddname as a DDEF command retrieved through this macro instruction.

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: If no ddnames are given, all DDEF commands in the set are retrieved and executed.

At completion of execution of the CDD macro instruction, the low-order byte of register 15 contains one of the following codes:

Code (Hexadecimal)	Significance
00	Normal
04	Invalid dsname
08	Invalid ddname
0C	ddname not in data set
10	Error return from DDEF
14	Not a line data set

L- AND E-FORM USE: The oplist operand is required in the L-form of this macro instruction and is not permitted in the E-form. Only the text form of the operand can be used in the L-form of the macro instruction.

DCBD -- Provide Symbolic Names for a Data Control Block (O)

The DCBD macro instruction generates a dummy control section (DSECT) that provides symbolic names for the fields in a data control block. With proper initialization of a base register, the user may access all fields of a data control block.

The following conventions have been adopted:

1. The name of the dummy control section is CHADCB. (An EQU is included in the DSECT to allow use of the alternative OS/360 name IHADCB).
2. The name of each field begins with the characters "DCB" followed by the keyword operand that represents the field in the DCB macro instruction. If the resulting name is longer than six characters, it is truncated to six characters by right-to-left dropout; that is, the field represented by the operand BLKSIZE= should be written DCBBLK. (Refer to Appendix F.)

The attributes of each data control block field are defined in the dummy control section (DSECT). Data control block fields containing addresses are aligned on fullword boundaries.

Name	Operation	Operand
	DCBD	

CAUTION: The DCBD macro instruction may be used only once in an assembly module.

PROGRAMMING NOTES: The macro instruction may appear at any point in a control section. However, if it is written at any location other than at the end of a control section, the original control section must be resumed by the user. The data control blocks to be accessed need not appear in the same control section as the DCBD macro instruction.

EXAMPLE: This example illustrates how a program can access a field in a data control block through use of the DCBD macro instruction. The load address (LA) instruction is used to place the address of the data control block in register 5.

A USING statement establishes a base register for CHADCB. The store operation (ST) places the value contained in register 6 into the specified field of the data control block pointed to by register 5. DCBLRE is the field associated with logical record length. The user previously loaded register 6 with the value he desired to be in DCBLRE.

```
          .
          .
          .
MYDCB     DCB         DDNAME=MYDCB,MACRF=G  (other DCB operands)
          .
          .
          .
          LA          5,MYDCB
          USING       CHADCB,5
          ST          6,DCBLRE
          .
          .
          .
          DCBD
```

FINDDS* - Locate JFCB Corresponding to Data Set Name (S)

The FINDDS macro instruction obtains the location of the JFCB corresponding to a given data set name. If the data set name specified is not in the task definition table (TDT), but is in the catalog, the user can request that a JFCB be created.

FINDJFCB* - Locate JFCB and Ensure Volume Mounting (S)

The FINDJFCB macro instruction locates the JFCB for a given data definition name and, optionally, ensures that the volumes specified in that JFCB are mounted.

*Although these macro instructions are available to all users, they are employed primarily by system programmers; therefore, refer to System Programmer's Guide for a discussion of these macro instructions.

CONNECTING A DATA SET TO THE SYSTEM

Before processing a data set, a user must first describe its attributes and then connect it to the system. User issuance of the OPEN macro instruction causes the system to interrogate the data set attribute information specified by the DDEF and DCB macro instructions or any other available sources for such attributes. The system determines if an appropriate data set organization has been specified and if all of the necessary attributes for processing such a data set have been provided. If the user has indicated he wants to alter the DCB contents at open time, by including the EXLST parameter (for BSAM and QSAM only) with his attribute specifications, the system immediately exits to the user modification routine. Once all the required attributes have been provided, the system makes the access method that a user has indicated he desires to employ (via attribute specifications) available to him. At the time a user opens a data set he can optionally select or default a processing option which indicates to the system the type of processing he expects to perform on that data set.

The user should know that the processing option he specifies when he issues the OPEN macro instruction determines whether he can use all of the macro facilities of an access method or only a portion of them (i.e., if a user opens a data set for INPUT only, he will only be allowed to use macro instructions which retrieve data and will not be allowed to use macro instructions that store data into the data set he has opened).

Once the system knows the processing option and locates the device on which a data set is to reside, or currently resides, it proceeds to physically open that data set by processing labels (if specified) and physically positioning the user at the data record he wants to process. The initial positioning directed by the system varies depending on the access method, the processing option, device type, and in some cases the status (i.e., MOD) of the data set. These relationships are described in detail in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032. The functions of the OPEN macro are briefly summarized below.

OPEN collects the attribute data, describing one or more data sets, from the various sources for providing such attributes (such as the DCB and DDEF macro instructions), by priority, and places them in the related data control blocks. These attributes are made available to the system, thereby logically connecting the data set(s) to the system. The access method dependent portion of the data set's data control block is initialized with pointers to the appropriate access method routines. Labels (if any exist) are checked, the user's privilege class is verified, and the system positions the user at the beginning of the data set that is to be processed. The user can proceed to process an opened data set.

A detailed explanation of the above macro instruction and the various formats in which it may be specified (depending on access method) is shown below. Further information pertaining to opening a data set and the priority of attribute sources may be found in Appendix F of this publication and in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

OPEN -- Connect a Data Set to the System (S)

The OPEN macro instruction connects one or more data sets to the system by completing the data control blocks containing their attributes, indicates the manner in which a data set is to be processed, and initially positions the data set for processing. Input labels are analyzed and output labels are created. Control is given to exit routines as specified in the data control blocks exit list (BSAM and QSAM only). Any number of data sets and their associated options may be specified in the OPEN macro instruction.

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

Name	Operation	Operand
[symbol]	OPEN	({dcb-addr,[(opt_1-code[,opt_2-code])]},...)

dcb
: specifies the address of the data control block containing the attributes of the data set that is to be initialized.

opt_1
: specifies the intended method of input/output processing of the data set being connected to the system. The processing method which can be specified is dependent on the data set organization and access method which is being used to perform the I/O processing. The various processing options, their meanings, and the access methods with which they can be specified are indicated below:

Code	Meaning	VAM	BSAM	QSAM	IOREQ
INPUT	Data set can be used as input only. This option is assumed if opt_1 is defaulted.	X	X	X	X
OUTPUT	Data set can be used for output only	X	X	X	X
INOUT	Both input and output operations are allowed. The DCB is opened as INPUT.	X	X	--	X
OUTIN	Both output and input operations are allowed. The DCB is opened as OUTPUT.	X	X	--	X
UPDAT		X	X	X	X
RDBACK	An INPUT data set is to be read backwards.	--	X	X	--

Note: Opening a VISAM data set for INOUT or OUTIN is equivalent to opening for UPDAT. When a data set is opened for UPDAT, however, the user must position to the desired record in the data set.

opt_2
: the codes REREAD and LEAVE are accepted for compatibility with the

IBM System/360 Operating System. However, this parameter is ignored by TSS/360 because volumes are not mounted in parallel.

CAUTION: The following errors cause the results indicated:

Error	Result
Opening a data control block that is already open	No action
Specifying the address of an invalid data control block	Task terminated.
Opening a data control block when a DDNAME in data control block has not been provided.	Nonconversational task terminated; prompting given if task is conversational
Opening a privileged data set by a nonprivileged user (BSAM, QSAM, VPAM and IOREQ only).	Task terminated
Opening a READ-ONLY data set and specifying an option other than INPUT	Task terminated
Opening a data control block when the DDNAME in the data control block does not correspond to the DDNAME in the DDEF macro instruction (or command)	Nonconversational task terminated; prompting given if task is conversational
Opening a data control block containing an invalid DSORG specification	Task terminated

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

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, which is more efficient than to open them individually. One of the services performed at this time is processing of labels of data sets or volumes.

VSAM:
When a shared VSAM data set is opened, a data set interlock is set according to the opt_4 specification. If INPUT is specified, a read interlock is set; if OUTPUT, INOUT, OUTIN, or UPDAT is specified, a write interlock. Rules for sharing VSAM data sets are given in Appendix K.

VISAM:
When a shared VISAM data set is opened, a data set interlock is set according to the opt_4 specification. If INPUT, INOUT, OUTIN, or UPDAT is specified, a read interlock is set; if OUTPUT is specified, a write interlock is set. Rules for sharing VISAM data sets are given in Appendix K.

BSAM:
If a DCB exit routine or a user-label exit routine is to be executed, the exit list address must be provided in the data control block. The format of the exit list, its use during the open-

ing process, and exit routine requirements are discussed in Appendix A.

L- AND E-FORM USE: The L- and E-form of this macro instruction are allowed. The E-form of the macro instruction may specify any parameters; however, the parameters specified in the E-form will overlay parameters specified in the L-form. The E-form may not specify more DCB operands than are specified in the L-form. The format of the parameter list generated by the OPEN macro instruction is described in Appendix L.

For example:

```
JOE     OPEN     (DATSET,,MORSET,,),MF=L
DEB     OPEN     (,,FOSET,,NUSEM),MF=(E,JOE)
```

When the E-form macro instruction is executed, the data control block FOSET replaces MORSET in the parameter list. Data control blocks with symbolic addresses DATASET, FOSET, and NUSEM are opened.

EXAMPLES: EX1 opens the data control block INVEN as an input data set. EX2 opens the two data control blocks INVEN and REPORT with different options. EX3 opens the two data control blocks INVEN and MASTER; they are opened for input data sets since INPUT is assumed when opt_1 is omitted. EX4 generates a parameter list for opening INVEN, and EX5 opens INVEN.

```
EX1     OPEN     (INVEN,(INPUT))
EX2     OPEN     (INVEN,(INPUT),REPORT,(OUTPUT,LEAVE))
EX3     OPEN     (INVEN,,MASTER)
EX4     OPEN     (INVEN,(INPUT)),MF=L
EX5     OPEN     MF=(E,EX4)
```

ACCESSING A DATA SET

Once a data set has been given a name, its attributes have been described, and it has been connected to the system, the user can employ the routines provided by the TSS/360 data set management facilities for storing and retrieving data organized in the various formats. These routines are employed by using I/O macro instructions in the user's source program. The macro instructions used comprise part of an access method and are dependent on the manner in which a user organizes and desires to process his data. There are two primary types of access methods, the Virtual Access Methods (VAM) and the Sequential Access Methods (SAM) as indicated below.

VAM: These are the access methods used in TSS/360 unless the data sets must be interchanged with programs running in Operating System/360 or the Model 44 Programming System, or the data set is to be written on magnetic tape.

Users create, read, and process Virtual Access Method (VAM) data sets on the basis of logical records. The system, however, blocks these records by pages (4096 bytes) and uses the page as the unit of transfer between the direct access device and the user's virtual storage. The system also ensures that only those pages of a data set that are actually required are resident in virtual storage. Because VAM data sets can be organized either sequentially, indexed sequentially, or partitioned, three distinct access methods are provided under VAM for processing these data sets. The virtual access methods that are provided to a user are:

Data Set Organization	Access Method
sequential	Virtual Sequential Access Methos (VSAM)
indexed sequential	Virtual Indexed Sequential Access Method (VISAM)
partitioned	Virtual Partitioned Access Method (VPAM)

SAM: Used to read and write records that can be read and written with programs running under control of the Operating System/360 or the Model 44 Programming System, or when the data set is to be written on magnetic tape.

Users create, read, and process SAM data sets on the basis of physical records. The records within a physical record can, however, be blocked or unblocked. Because of this, two distinct access methods are provided under SAM for processing data sets. The Sequential Access Methods are indicated below.

Data Set Organization	Access Method
unblocked sequential	Basic Sequential Access Method (BSAM)
blocked sequential	Queued Sequential Access Method (QSAM)

Another special accessing facility, the Input/Output Request Facility (IOREQ) is provided for users who would rather program their own I/O device control routines than employ any of the access methods provided with the TSS/360 Data Management Facilities.

Each of the above access methods and the macro instructions which may be used with them are explained more fully on the following pages. Detailed information pertaining to access methods and data set organization may be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

VIRTUAL SEQUENTIAL ACCESS METHOD

The virtual sequential access method (VSAM) consists of the TSS/360 data management facilities that enable a user to process virtual sequential data sets. These data sets can be stored on, or retrieved from, direct-access devices only. The record format within each such data set can be fixed length (blocked or unblocked), variable length (blocked or unblocked), or undefined length (unblocked only). Such attributes are unique for each data set; they must be defined to the system before a data set can be accessed by VSAM. The macro instructions that have been provided to a user, by VSAM, for accessing a data set in the appropriate manner, are summarized below.

GET used for reading logical records in a sequential order.

PUT for writing new or altered logical records into a virtual sequential output data set.

PUTX for writing an updated or identical logical record, directly from an input data set to an output data set, without altering the length of the record. The next sequential logical record contained in an input buffer area (where it may have been modified) is transferred to the output buffer as the next sequential output record. The system must be positioned at that next sequential logical record by issuing a locate mode GET macro instruction prior to issuing PUTX.

SETL enables a user to logically position a data set at its beginning, end, at the previous record, or at any logical record within a blocked sequential data set. Subsequent PUT or GET operations will start at this new position.

Detailed explanations of each of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to BSAM data set management and the related macro instructions can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

GET -- Get a Record (R)

The GET macro instruction (for VSAM) can be specified in either locate mode or move mode. In locate mode, the GET macro instruction locates the next sequential record of an input data set and places its address in register 1. The user may then operate on the record where it is, or move it to a work area. In move mode, the GET macro instruction acquires the next sequential record of an input data set and moves it to a specified area in virtual storage.

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 (for move mode only)
specifies the address of the user's work area into which the record is moved. If (0) is written, the address must be loaded into parameter register 0 before execution of this macro instruction.

CAUTION: If a GET macro instruction is requested beyond the end of a data set, as a result of sequential operation or following a SETL macro instruction, the user EODAD is given control. (Refer to Appendix C.)

The address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: When retrieving variable-length records, the GET macro instruction returns with the length of the logical record in the DCBLRE field of the data control block.

For undefined-format records, the user must set the DCBLRE field to the length of the record to be retrieved before issuing GET.

Rules for sharing VSAM data sets are given in Appendix K.

PUT -- Include a record in an Output Data Set (R)

The PUT macro instruction (for VSAM) can be specified in either locate mode or move mode. In locate mode, the PUT macro instruction places in register 1 the address of an output buffer. The user should subsequently construct at that address the next record to be incorporated in an output data set. In move mode, the PUT macro instruction moves a record from a user-specified area in virtual storage into an output buffer so that the system may include the record in the output data set.

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 (for move mode only)
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.

CAUTION: The address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: It is the user's responsibility to store the length of each format-U record in the DCBLRE field of the data control block before issuing the PUT. This length must be a multiple of 4096 bytes.

For format-V records, each record includes four control bytes. The user must store the length of the record in bytes 1, 2, and 3 of that four-byte field, before issuing a PUT macro instruction. Byte 0 must contain binary zero.

Rules for sharing VSAM data sets are given in Appendix K.

PUTX -- Replace a Sequential Logical Record (R)

The PUTX macro instruction (for VSAM) allows the user to return an updated logical record to an input 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.

CAUTION: The address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: The PUTX macro instruction can only replace a record that was located by a locate-mode GET macro instruction. The data control block must be opened for the UPDAT mode while using PUTX. The user must not change the length of the record during the replacement process.

Rules for sharing VSAM data sets are given in Appendix K.

SETL -- Specify Start of Sequential Processing (R)

The SETL macro instruction (for VSAM) positions to the beginning, end, previous record, or any point within a virtual sequential data set.

Name	Operation	Operand
[symbol]	SETL	dcb-{addrx | (1)},type-{R | B | E | P}[,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 of the data control block must be in register 1 before execution of this macro instruction. type
specifies the starting point for processing and any optional services requested:

R
Record at the retrieval address obtained from DCBLPDA field in the data control block following a GET or PUT.

B
Beginning of the data set

E
End of the data set

P
Previous logical record (backspace)

llimit
specifies the address of a word containing the retrieval address. If the type operand specifies B, P, or E, the llimit field is to be omitted. If (0) is written, the address of a field containing the retrieval address must be in register 0 before execution of this macro instruction.

CAUTION: A backspace request is not permitted for format-U records and causes abnormal termination.

The address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: A SETL instruction that positions to an area outside of the data set causes an error. The error is indicated during a subsequent GET or PUT macro instruction by exit to EODAD.

Rules for sharing VSAM data sets are given in Appendix K.

VIRTUAL INDEXED SEQUENTIAL ACCESS METHOD

The virtual indexed sequential access method (VISAM) consists of the TSS/360 data management facilities that enable a user to process indexed sequential data sets. These data sets may be stored on, or retrieved from, direct access devices only. The record format within each such data set can be fixed-length (blocked or unblocked) or variable-length (blocked or unblocked) format. Such attributes are unique for each data set; they must be defined to the system before a data set can be accessed by VISAM. The macro instructions that have been provided to a user by VISAM, for accessing a data set in the appropriate manner, are indicated below.

GET for reading logical records in sequential order

PUT for writing logical records in a sequential order

READ for reading logical records in a nonsequential or sequential order

WRITE for writing logical records in a nonsequential or sequential order

SETL enables a user to logically position a data set at its beginning, end, at the previous record, or at any logical record within a blocked sequential data set. Subsequent PUT or GET operations will start at this new position.

ESETL (for shared data sets) allows other sharers to access portions of the data set currently being processed by the user.

RELEX (for shared data sets) allows other sharers to access and/or update portions of the data set currently being processed by the user.

DELREC deletes a specified logical record from a data set

Detailed explanations of each of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to VISAM data set management and the related macro instructions can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

GET -- Get a Record (R)

The GET macro instruction (for VISAM) can be specified in either locate mode or move mode. In locate mode, the GET macro instruction locates the next sequential record in an input data set and places its address in register 1. The user may then operate on the record where it is, or move it to a work area. In move mode, the GET macro instruction acquires the next sequential record and moves it from an input buffer to a user-specified area in virtual storage.

Name	Operation	Operand
[symbol]	GET	dcb-{addr / (1)} ,area-{addrx / (0)}

dcb
specifies the address of the data control block opened for the

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

area (for move mode only)
specifies the address of the user's work area into which the record is to be moved. If (0) is written, the address must have been loaded into parameter register 0 before execution of the macro instruction.

CAUTION: Any exceptional condition (i.e., logical record out of sequence) resulting from the execution of a GET macro instruction causes control to be passed to the user's synchronous error exit (SYNAD) routine. In this case, the general registers and the exceptional condition fields in the data control block are set as shown in Appendixes B and F.

The address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: When retrieving variable-length records, the GET macro instruction returns with the length of the logical record in the DCBLRE field of the data control block.

If a GET is requested beyond the end of a data set, as a result of sequential operation or SETL macro instruction, the user EODAD exit is taken. See Appendix C.

A page-level read interlock is imposed on the page referred to by execution of this macro instruction. The interlock is released by any macro instruction referring to the same DCB that refers to another page. Rules for sharing VISAM data sets are given in Appendix K.

PUT -- Include a Record in an Output Data Set (R)

The PUT macro instruction (for VISAM) may be specified in either locate mode or move mode. In locate mode, the PUT macro instruction places in register 1 the address of an output buffer. The user should subsequently construct, at this address, the next record for incorporation into the output data set. In move mode, the PUT macro instruction moves a record from a specified area in virtual storage to an output buffer so that the system can include the record in the output data set.

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 (for move mode only)
specifies the address of the 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.

CAUTION: Any exceptional condition resulting from the execution of a PUT macro instruction causes control to be passed to the user's synchronous error exit (SYNAD) routine. In this case, the general registers

and the exceptional condition fields in the data control block are set as shown in Appendixes B and F.

The address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: For format-V records, each record must begin with a four-byte length field. The user must place the length of the record into the low-order three bytes of that four-byte field, before issuing a PUT macro instruction. The high-order byte must contain binary zero. The PUT macro instructions may not be used with a shared data set. Rules for sharing VISAM data sets are given in Appendix K.

READ -- Read a Selected Logical Record (S)

The READ macro instruction (for VISAM) acquires a selected logical record from an input data set and moves it to a user-specified area. The user selects the record by providing either the record key or the retrieval address. The key is in the user's data control block upon completion of the read operation; when completed, processing of the user's program continues.

Name	Operation	Operand
[symbol]	READ	decb-symbol,type-{KY / KZ / KX},dcb-addr,area-addr,key-addr

decb
: specifies the symbol (name) to be assigned to the data event control block (DECB) constructed as part of the expansion of this macro instruction.

type
: specifies one of the following as the type of READ operation.

  KY - read according to specified key.

  KZ - read according to specified retrieval address.

  KX - read according to specified key permitting no other user sharing the data set to gain access to the record until the current user has released the record. The record must be released by the RELEX macro instruction or by a subsequent WRITE macro instruction referring to the same data control block.

dcb
: specifies the address of the data control block opened for the data set being processed.

area
: specifies the address of the user's work area into which the record will be placed.

  Note: The area must be large enough to contain the largest expected record.

key
: specifies the address of the field containing either the record key for a READ (type -KY or -KX) or the retrieval address for a READ

(type KZ). The retrieval address is a four-byte field, beginning on a word boundary that is in the data control block and may be accessed using the DCBD macro instruction and the name, DCBLPA.

CAUTION: Exceptional conditions, including "key not found," "key greater than last key on data set," and "invalid retrieval address," resulting from the execution of a READ macro instruction, cause control to be passed to the user's synchronous error exit (SYNAD) routine. In this case, the general registers and the exceptional condition fields in the data control block are set as shown in Appendixes B and F.

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: READ (type KY) imposes a page-level read interlock on the pages containing the record to be read whereas READ (type KX) imposes a page-level write interlock and releases a page-level read interlock. As the record pointed to by the data control block shifts within the data set, page-level interlocks are released from pages no longer being used. The retrieval address form of READ (i.e., type KZ) cannot be used with shared data sets.

Rules for sharing VISAM data sets are given in Appendix K.

L- AND E-FORM USE: The L-form macro instruction results in a macro expansion consisting of only a parameter list (DECB). The format of the DECB is described in Appendix B.

The E-form macro instruction results in a macro expansion consisting of only executable instructions. The E-form macro instruction uses the DECB built for it by the L-form macro instruction. Only MF=E should be specified for the MF= operand of the E-form, because it is the DECB symbol which names the parameter list of the L-form.

If the E-form is used, either a DECB symbol or (1) must be specified; if (1) is specified, the address of a DECB must be loaded into register 1 before execution of this macro instruction. Any E-form parameter replaces the corresponding specified optional or required parameter in the DECB. If a parameter is not specified in the L-form, it must be specified in the E-form. Certain required parameters for the E- and L-form are:

Operand	L-Form	E-Form
decb	required	decb-{symbol / (1)}
type	required	required
MF	MF=L	MF=E

WRITE -- Write a Selected Record (S)

The WRITE macro instruction (for VISAM) moves a selected record from a user-specified area to an output buffer. The system then includes the record in the output data set either by key or retrieval address. This macro instruction may be used to update a record or add to the data set. When the write operation is completed, processing of the user's program continues.

Name	Operation	Operand
[symbol]	WRITE	decb-symbol,type-{KR / KS / KT},dcb-addr,area-addr,key-addr

decb
specifies the symbol (name) to be assigned to the data event control block (DECB) constructed as part of the expansion of this macro instruction.

type
specifies one of the following as the type of WRITE operation:

KR - WRITE replace by retrieval address
KS - WRITE replace by key
} for updating

KT - WRITE a record with a new key } for adding a record

dcb
specifies the address of the data control block opened for the data set being processed.

area
specifies the address of the user's work area from which the record is to be written.

key
specifies the address of the field containing either the record key, the length of which is indicated in the data control block; or a retrieval address, a four-byte field on a fullword boundary, originally obtained from DCBLPA.

CAUTION: Exceptional conditions resulting from the execution of a WRITE macro instruction cause control to be passed to the user's synchronous error exit (SYNAD) routine. In this case, the general registers and the exceptional condition fields in the data control block are set as shown in Appendixes B and F.

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: WRITE releases any page-level interlocks set for the data set as a result of executing macro instructions referring to the same data control block. Rules for sharing VISAM data sets are given in Appendix K.

L- AND E-FORM USE: The L-form macro instruction results in a macro expansion consisting of only a parameter list (DECB). The format of the DECB is described in Appendix B.

The E-form macro instruction results in a macro expansion consisting of only executable instructions. The E-form macro instruction uses the DECB built for it by the L-form macro instruction. Only MF=E should be written for the MF= operand in the E-form, because it is the DECB symbol which names the parameter list of the L-form.

If the E-form is used, either a DECB symbol or (1) must be specified. If (1) is specified, the address of a DECB must be loaded into register 1 before execution of this macro instruction. Any E-form parameter

replaces the corresponding specified optional, or required parameter in the DECB. If a parameter is not specified in the L-form, it must be specified in the E-form. Certain required parameters for the E and L-form are as follows:

Operand	L-Form	E-Form
decb	required	decb-{symbol / (1)}
type	required	required
MF	MF=L	MF=E

SETL -- Specify Start of Sequential Processing (R)

The SETL macro instruction (for VISAM) positions a data set to the beginning, end, previous record, or any point within the data set.

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 data control block address must have been loaded into parameter register 1 before execution of the macro instruction.

type
specifies positioning within the data set as follows:

Code	Positioning
R	Record at the retrieval address obtained from the DCBLPA field in the data control block following a GET or PUT
B	Beginning of the data set
E	End of the data set
P	Previous record (backspace)
K	Record whose key is specified in the operand
N	Record immediately following the one pointed to by the previous SETL; if there was no previous SETL, no repositioning occurs

llimit
specifies the address of a field containing either the record key, the length of which is indicated in the data control block, or a retrieval address (a four-byte field beginning on a fullword boundary originally obtained from DCBLPA). If (0) is written, the llimit address must have been loaded into parameter register 0 prior

to execution of this macro instruction. If the type operand is specified as B, P, N, or E, the llimit field is ignored.

CAUTION: Exceptional conditions, including the following three conditions, resulting from the execution of a SETL macro instruction cause control to be passed to the user's synchronous error exit (SYNAD) routine. In this case, the general registers and the exceptional condition fields of the data control block are set as shown in Appendixes B and F.

1. Invalid retrieval address or record key.
2. SETL (N) following a SETL (E).
3. SETL (P) following a SETL (B).

If a SETL macro instruction is requested by key and the request key is greater than the highest key or lower than the lowest key in the data set, control is passed to the user's SYNAD routine. SETL by retrieval address (type R) must not be used with a shared data set.

The address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTE: SETL does not impose any sharing interlocks on a data set. Rules for sharing VISAM data sets are given in Appendix K.

ESETL -- Release Shared Data Set (R)

The ESETL macro instruction (for VISAM) releases a page-level interlock imposed by another macro instruction (e.g., GET, or READ). This macro instruction does not release the write interlock caused by a type KX READ. See RELEX macro instruction in this section.

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 data control block address must have been loaded into parameter register 1 before execution of this macro instruction.

CAUTION: Exceptional conditions resulting from the execution of a ESETL macro instruction cause control to be passed to the user's synchronous error exit (SYNAD) routine. In this case, the general registers and the exceptional condition fields in the data control block are set as shown in Appendixes B and F.

PROGRAMMING NOTE: Rules for sharing VISAM data sets are given in Appendix

DELREC -- Delete a Record (R)

The DELREC macro instruction (for VISAM) deletes a specified record from a virtual index sequential data set. The record may be specified by its key or its retrieval address.

Name	Operation	Operand
[symbol]	DELREC	dcb-{addrx / (1)},type-{K / R},llimit-{addrx / (0)}

dcb
specifies the address of the data control block opened for the dataset being processed. If (1) is written, the address must be loaded into parameter register 1 prior to execution of this macro instruction.

type
specifies whether the record will be deleted by key or retrieval address as follows:

K
Record key

R
Retrieval address as obtained by the user from DCBLPA in the data control block.

llimit
specifies the address of a field containing either the record key or the retrieval address. The retrieval address must be in a four-byte field, beginning on a doubleword boundary. If (0) is written, the address must be loaded into parameter register 0 prior to execution of the macro instruction.

CAUTION: Exceptional conditions, including "invalid retrieval address" and "key not found," resulting from the execution of a DELREC macro instruction cause control to be passed to the user's synchronous error exit (SYNAD) routine. In this case, the general registers and the exceptional condition fields of the data control block are set as shown in Appendixes B and F. DELREC by retrieval address may not be used with a shared data set.

PROGRAMMING NOTE: This macro instruction releases any page-level interlocks established by other macro instructions referring to the same DCB. Rules for sharing VISAM data sets are given in Appendix K.

RELEX -- Release Read Exclusive Record (R)

The RELEX macro instruction (for VISAM) makes a record of a shared data set available to other users after the record has been read with a READ exclusive (type KX) macro instruction.

Name	Operation	Operand
[symbol]	RELEX	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 of the data control block must be loaded into parameter register 1 prior to executing the macro instruction.

CAUTION: Exceptional conditions resulting from the execution of a RELEX macro instruction cause control to be passed to the user's synchronous error exit (SYNAD) routine. In this case, the general registers and the exceptional condition fields in the data control block are set as shown in Appendixes B and F.

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address ofa save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTE: Rules for sharing VISAM data sets are given in Appendix K.

VIRTUAL PARTITIONED ACCESS METHOD

The virtual partitioned access method (VPAM) consists of the TSS/360 data set management facilities that enable a user to access partitioned data sets. Each partitioned segment (or member) is a complete VSAM or VISAM data set in itself. The allowable organizations of the records within members are the same as within VSAM or VISAM respectively. VPAM may be used only to store or retrieve data set members on direct access devices.

Once a partitioned data set has been defined and connected to the system by previous user (or system) issuance of a DCB, DDEF, and OPEN macro instruction the user may employ the VPAM macro instructions (FIND and STOW) to locate its members. When the member is opened and located via a FIND macro instruction, the macro instructions, appropriate to the particular member's organization (i.e., VSAM or VISAM), can be used to process the member. It should be noted that although a member is defined by the same DDEF and DCB macro instructions that defined the partitioned data set, the member is not opened until a VPAM FIND macro instruction is executed. The VPAM macro instructions are briefly described below.

FIND locates an individual member within a VPAM data set and opens the member for processing. To process the records within the member, appropriate VISAM and VSAM macro instructions can be employed.

STOW causes a VISAM or VSAM data set, previously defined to the system as a partitioned data set member, to be incorporated or deleted from a partitioned data set. It also adds, changes, deletes, or replaces member names or aliases and allows a user to enter unique data, describing the member, into an index.

Detailed explanations of each of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to VPAM data set management and the related macro instructions can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

FIND -- Find a Member of a Partitioned Data Set (S)

The FIND macro instruction (for VPAM) searches a partitioned organization directory to locate a directory entry for a member and optionally places the user's data associated with the member into the specified area. The member is opened and positioned for processing.

Name	Operation	Operand
[symbol]	FIND	dcb-addr,name-addr[,area-addr,length-value]

dcb
specifies the address of the data control block opened for the data set being processed.

name
specifies the location of the eight-character member name, or alias, that is to be used to locate the member.

area
specifies the location of the eight-character member name, or alias, that is to be used to locate the member.

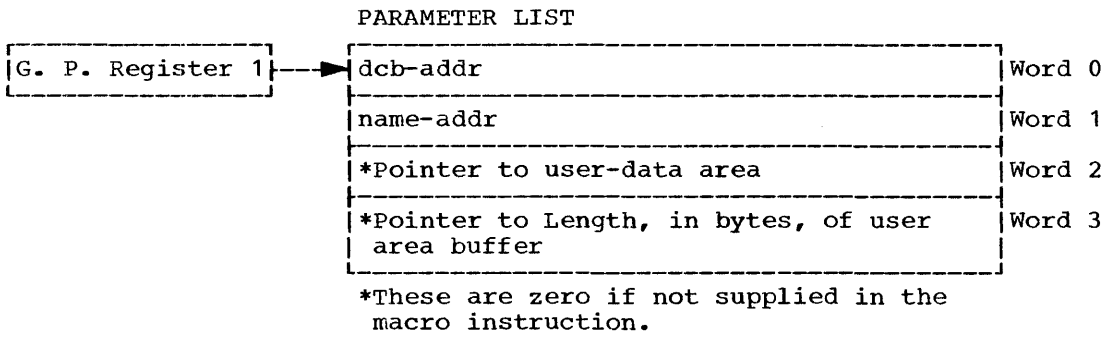
length
specifies the length, in bytes, of the area provided for reading in the user data.

CAUTION: If area is specified, length must be specified. In addition, area and length must be specified for shared data sets if user data is present. If not specified, the task is abnormally terminated.

The FIND macro instruction causes an abnormal termination if any conditions are discovered that make continuation impossible.

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: After execution of the FIND macro instruction, general register 0 contains the length of the user data in the POD. General register 1 points to the parameter list shown below.



The length, in bytes, of the user area buffer is placed in a word immediately following word 3 of the parameter list by the macro expansion. However, if the user constructs his own parameter list, the word containing this length may be placed in some other location.

If the length specified is less than the actual length of the user data in the POD, both area and length operands are ignored and general register 15 contains appropriate error code (hexadecimal 10).

Rules for sharing VPAM data sets are given in Appendix K.

For shared VPAM data sets, the following interlocks are set by a FIND macro instruction:

1. VISAM members are:
   - write interlocked when opened for OUTPUT.
   - read interlocked when opened with any other option.
2. VSAM members are:
   - read interlocked when opened for INPUT.
   - write interlocked when opened with any other option.

After execution of the FIND macro instruction, bits 24 through 31 of general register 15 contain one of the following codes, indicating the status of the operation. The user should take appropriate action depending on the code returned.

Code (Hexadecimal)	Definition
00	Successful completion of FIND
04	Member or alias was not located by FIND
08	Data control block, indicated in the macro instruction is in use for creating a member. Execution of a STOW must be complete before this FIND can be executed
10	Length specified in the macro instruction is not large enough to contain user data
14	Member to be located is already open for this data control block, due to previous FIND

L- AND E-FORM USE: All operands are optional in the L-form of this macro instruction; register notation may not be used. All operands are optional in the E-form; register notation may be used. All operands not supplied in the L-form must be supplied in the E-form.

STOW -- Manipulate Partitioned Organization Directory (R)

The STOW macro instruction (for VPAM) causes a partitioned data set member to be incorporated or deleted from a partitioned data set. This macro instruction is also used to add, change, delete, or replace a member name or an alias. It also provides for storage of additional information in the partitioned organization directory (POD) in the form of user data.

Name	Operation	Operand
[symbol]	STOW	dcb-{addrx / (1)}, [area-{addrx / (0)}], type-code

dcb
: specifies the address of the data control block opened for the data set being processed. If (1) is written, the address must be loaded into parameter register 1 before execution of this macro instruction.

area
: specifies the address of an area constructed by the user. The contents of this area depend on the type of STOW requested. (Refer to "Programming Notes.") If (0) is written, the address must be loaded into parameter register 0 before execution of this macro instruction. For type-R STOW, area does not have to be specified, and if not specified, the original user data will be unchanged.

type
: specifies the type of STOW being requested by one of the following codes:

  N Add a new member and close the member.

  NA Add one or more new aliases.

R Replace the user data associated with a member and close the member.

U Replace the user data associated with a member but do not close the member.

D Delete a member from the data set; the directory entries for the member and all of its aliases are deleted and the space occupied by the member is made available for subsequent use.

DA Delete one or more aliases.

C Change the name of a member.

CA Change the name of an alias.

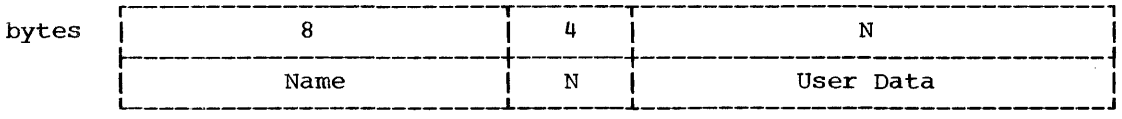
CAUTION: A member may not be subsequently referred to by the same data control block after a type-N or -R STOW until a FIND of that member is again requested since these types of STOW close the member.

STOW abnormally terminates the task if any conditions are discovered that make continuation impossible.

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: Only type-R STOW is permitted on a shared data set opened for input. The format of the area used by the STOW macro instruction depends on the type of STOW requested. It is the user's responsibility to construct the area and pass the address of the area to STOW in the area operand of this macro instruction. The area requirements are:

Types N and U: The area must be at least 12 bytes long and begin on a fullword boundary.

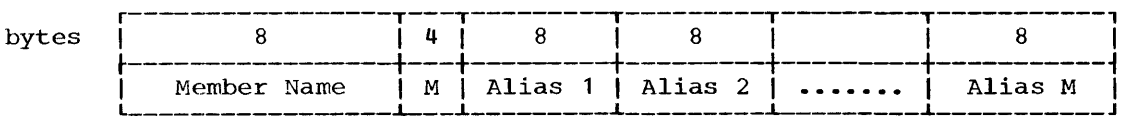


Name - Eight-character member name

N - Number of bytes of user data (0≥N≥510)

User Data - Contains the variable data supplied by the user. The data are stored in the POD and can be retrieved by means of the FIND macro instruction.

Types NA and DA: The area must be at least 20 bytes long and begin on a fullword boundary.



Member Name - Name of the member to which the aliases are linked or are to be linked.

M - Number of aliases to be added or deleted.

Aliases - The aliases to be added or deleted.

Type D: The specified area must contain the member name that is to be deleted. It is eight bytes long. When a member name is deleted, all of its aliases are also deleted.

bytes	8
	Member Name

Type C: The name of the member and the name to which it is to be changed are in this area (16 bytes).

bytes	8	8
	Member Name	New Member Name

Type CA: The area specified must be 24-bytes long.

bytes	8	8	8
	Member	Old Alias	New Alias

Member - The eight-character name of the member with which the old alias is associated.

Old Alias - The eight-character alias being changed.

New Alias - The eight-character alias being used for the replacement.

Type R: If any user's data is specified, the length must be four bytes longer than the length of the data and begin on a fullword boundary. The additional four bytes are required to specify the length of the specified data.

bytes	4	N
	N	User Data

N - Number of bytes of user's data to be placed in the POD ($0 \leq N \leq 510$)

User Data - Contains the variable data supplied by the user. The data is stored in the POD, and can be retrieved by means of the FIND macro instruction.

The user must have exclusive access to a member in order to issue type-C or type-D STOW; that is, he must have opened the data set with an OPEN option that causes the member to be write-interlocked.

Member interlocks are released by CLOSE (referring to the same DCB that caused the interlock to be set), type-R STOW, or a subsequent FIND.

Rules for sharing VPAM data sets are also given in Appendix K.

After execution of the STOW macro instruction, bits 24 through 31 of general register 15 contain one of the following codes indicating the status of the operation. The user should examine this code to determine the course of action.

Code (hex)	Definition
00	Successful completion of STOW
04	New name or alias is already in use (N, NA, C, or CA)
08	Member name is not in POD (U, D, DA, or CA)
10	Old member name is not in POD (C); alias is not in POD (DA); old alias is not in POD (CA)
14	Invalid type STOW requested

BASIC SEQUENTIAL ACCESS METHOD

The basic sequential access method (BSAM) consists of the TSS/360 data set management facilities that enable a user to access unblocked physical sequential data sets. Since BSAM does not provide a user with blocking/deblocking or buffering routines it should be used primarily to process unblocked records (QSAM has been provided to facilitate the processing of blocked records). A physical sequential data set can be stored on, or retrieved from, disk, tape, or cards, and can be printed out by a printer. The record format within each such data set can be fixed-length (blocked or unblocked), variable-length (blocked or unblocked), or undefined-length (unblocked only). Such attributes are unique for each data set; they must be defined to the system before a data set can be accessed by BSAM. The macro instructions provided to a user, by BSAM, for accessing a data set in the appropriate manner, are indicated below.

READ causes a request for a transfer of a physical record, from an I/O device directly to a specific virtual storage input area, to be recorded in a control block (DECB) and placed on an I/O request queue. Control is then returned to the user's program; the request is subsequently executed by the system when the device is available. If this physical record contains several logical records, the user must create his own deblocking routines to access the individual logical records. In such a case, the GETPOOL and GETBUF macro instructions are very useful.

WRITE causes a request for a transfer of a physical sequential record, from a specific storage area to an I/O device (directly, without using a buffer area), to be recorded in a control block (DECB) and placed on an I/O request queue. Control is then returned to the user's program and the request is subsequently executed by the system when the device is available. If a physical record is to contain several logical records, the programmer must write his own blocking routines to include the logical records in the storage area being transferred.

CHECK checks the queue of control blocks (DECBs) containing the requests for read or write operations to determine if those requests have been satisfied. It also indicates whether errors or exceptional conditions have occurred while satisfying the request.

DQDECB removes all unchecked DECBs (i.e., created by issuing READ and WRITE macro instructions) from a queue of unchecked DECBs maintained by the system.

GETPOOL requests allocation of an area in virtual storage for use as a buffer pool and assigns that area to a data control block describing the data set.

GETBUF obtains a buffer work area from a buffer pool previously assigned to a data control block (either by a GETPOOL macro instruction or as a result of having selected the buffer options provided in the DCB macro instruction).

FREEBUF returns a buffer work area, obtained by a GETBUF, to its buffer pool

FREEPOOL releases areas previously assigned to specified data control blocks as buffer pools (either by a GETPOOL macro instruction or as a result of buffer options specified by the DCB macro instruction).

BSP backspace one physical record or block on the current magnetic tape or direct access volume regardless of the direction in which data is being stored or retrieved on that device.

CNTRL provides a control for card stacker selection, printer carriage control, and magnetic tape positioning.

PRTOV controls the page format for an on-line printer by testing channels 9 and 12 on the printer control tape, as overflow indicators, and allows the user to provide an overflow subroutine to reposition the printer at any desired channel on the printer control tape.

FEOV advances the system to the next volume of a multivolume data set before the physical end of the current volume is reached.

POINT causes repositioning of tape or direct access volumes to a specified block within a data set on that device.

NOTE makes available to the problem programmer the relative position within a volume of a block just read or written.

Detailed explanations of each of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to BSAM data set management and the related macro instructions can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

READ -- Read a Block (S)

The READ macro instruction (for BSAM) transmits a block of data from an input data set to a user-specified virtual storage area. To allow overlap of the I/O operation with processing, the READ macro instruction returns control to the user's program before the input operation is complete.

The READ macro instruction may be used to read backwards from magnetic tape.

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 is illustrated in Appendix B, Table 7.

type
specifies one of the following:

SF
sequential forward reading of a physical sequential data set.

SB
sequential backward reading from a magnetic tape.

dcb
specifies the address of the data control block opened for the data set being processed.

area
specifies the address of an area in virtual storage into which the block of data is to be read. If SF is written in the type field, this operand specifies the address of the first byte of the area; if SB is written, the address of the last byte is specified.

length
specifies, for format-U records, the number of bytes to be transmitted. If 'S' is written, the program attempts to read the maximum size specified in the data control block, with maximum blocksize of 32,767 bytes. If this parameter is specified for format-F or format-V records, it is ignored. For format-F and -V blocks, length is obtained from the BLKSIZE field of the data control block.

CAUTION: Abnormal termination occurs if:

1. The specified data control block is not validly opened.
2. The specified DECB is already in use by a previous READ or WRITE macro instruction; i.e., it has not been checked by a CHECK macro instruction.
3. An attempt is made to issue a READ macro instruction that causes the number of unchecked READ and WRITE macro instructions to exceed the DCBNCP parameter specified in the data control block.
4. An attempt is made to read on a device that cannot execute the request, such as read the printer.
5. An attempt is made to read an OUTPUT data set.

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: The READ macro instruction returns control to the user's program before the transmission of data has been completed. To determine whether the read operation is 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 must not be reused or modified until the CHECK macro instruction is issued.

After a read operation has been checked, the length of a format-U block or a truncated block in a fixed-length blocked data set can be determined from the count field of the Channel Status Word in the DECB. The number of READs may not exceed that specified in the DCBNCP field in the data control block without using a CHECK macro instruction.

A data set written on a direct-access device with track overflow specified must have track overflow specified for all reads referring to that data set. If a track selected by a READ macro instruction is flagged as defective, the alternate track is automatically selected. For any device, the operator is notified if any intervention is required to complete the operation.

If a READ (type SB) macro instruction is issued for a format-V record, the address of the first byte of the record can be calculated by subtracting the count field in the channel status word from the maximum block size and subtracting the result from the area address.

If the length specified in the READ macro instruction for format-U records is less than the length of the actual physical record, the extra bytes of data are not transmitted.

The first four bytes on format-V blocks contain control information passed with the record, when read. The area specified by the area operand must be large enough to accommodate the maximum record size.

L- AND E-FORM USE: The L-form macro instruction results in a macro expansion consisting of only a parameter list (DECB). The format of the DECB is described in Appendix B.

The E-form macro instruction results in a macro expansion consisting of only executable instructions. The E-form macro instruction uses the DECB built for it by the L-form macro instruction.

If the E-form is used, either a DECB symbol or (1) must be specified; if (1) is specified, the address of a DECB must be loaded into register 1 before execution of this macro instruction. Any E-form parameter replaces the corresponding specified optional or required parameter in the DECB. If a parameter is not specified in the L-form, it must be specified in the E-form. Certain required parameters for the E- and L-form are:

Operand	L-Form	E-Form
decb	required	decb-{symbol / (1)}
type	required	required
MF	MF=L	MF=E

EXAMPLE: In example 1, a DECB, with the symbolic name ADECB, is produced as part of the in-line expansion. It indicates that forward-reading of the next block in the data set associated with data control block INDCB should be performed using area INAREA. The length operand, not written in this example, is required for format-U records.

In example 2, the type operand indicates backward reading of a block of records from the data set associated with the data control block INDCB. For format-U records, 100 bytes are transmitted from INAREA+99 to INAREA. If 'S' is specified for the length, the maximum block size is transmitted. For records other than format-U, the length parameter is ignored.

```
EX1     READ     ADECB,SF,INDCB,INAREA
EX2     READ     ADECB,SB,INDCB,INAREA+99,100
```

WRITE -- Write a Block (S)

The WRITE macro instruction writes (for BSAM) a block of data from virtual storage on a physical sequential data set. To allow overlap of the I/O operation with processing, the WRITE macro instruction returns control to the user's program before the output operation is complete.

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. (Refer to Appendix B, Table 7 for an illustration 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 processed.

area
: specifies the starting address of the area in virtual storage that contains the block of data to be written. The user must construct the record-length information in front of each block of format-V records.

length
: specifies, for format-U records, the number of bytes to be transmitted. If 'S' is written, the maximum block length (specified in the data control block) for the data set is used. If this parameter is specified for format-F or format-V records, it is ignored. For format-F blocks, the length value is obtained from the DCBBLK field of the data control block. For format-V blocks the length value is obtained from the first two bytes of the output area (LL).

CAUTION: Abnormal termination occurs if:

1. A WRITE macro instruction is issued with record length longer than a track, unless track overflow is specified in the DCB macro instruction.

2. The data control block specified is not validly opened.

3. The DECB specified is already in use by a previous READ or WRITE macro instruction; i.e., it has not been checked.

4. An attempt is made to issue a WRITE macro instruction which causes the number of unchecked READ and WRITE macro instructions to exceed the DCBNCP parameter in the data control block.

5. An attempt is made to write on a device which cannot execute the required operation,e.g., write on the card reader.

6. An attempt is made to write a data set opened for INPUT or RDBACK.

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: The WRITE macro instruction returns control before actual transmission of data is completed. To determine whether a write operation has been completed, the CHECK macro instruction must be issued for that DECB. The DECB employed for the write operation and the virtu-

al storage the block occupies must not be altered or used until the CHECK macro instruction is issued for that DECB.

If a track selected by a WRITE macro instruction is flagged as defective, an alternate track is automatically utilized. For any device, the operator is notified automatically if any intervention is required to complete the operation.

If the data set has been opened for UPDAT, the following considerations apply.

- 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 UPDAT option. Only the most recently read block can be updated and returned.

- The update mode is provided only for data sets on direct-access devices. Although 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 in storage
    .
    .
    .
WRITE                   Block A
    .
    .
    .
CHECK                   Await completion of write
```

  Thus, only the block last read, or its replacement, can be returned to the data set. Two READ macro instructions can be issued without an intermediate WRITE; this causes the first block to remain unchanged on the device.

L- AND E-FORM USE: The L-form macro instruction results in a macro expansion consisting of only a parameter list (DECB). The format of the DECB is described in Appendix B.

The E-form macro instruction results in a macro expansion consisting of only executable instructions. The E-form macro instruction uses the DECB built for it by the L-form macro instruction. Only MF=E should be written for the MF= operand in the E-form, because it is the DECB symbol which names the parameter list of the L-form.

If the E-form is used, either a DECB symbol or (1) must be specified. If (1) is specified, the address of a DECB must be loaded into register 1 before execution of this macro instruction. Any E-form parameter replaces the corresponding specified optional or required parameter in the DECB. If a parameter is not specified in the L-form, it must be specified in the E-form. Certain required parameters for the E and L forms are:

Operand	L-Form	E-Form
decb	required	decb-{symbol / (1)}
type	required	required
MF	MF=L	MF=E

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 for a format-U record, but for formats-V or -F, the length parameter is ignored.

```
EX1     WRITE     ADECB,SF,DCBOUT,AREA,800
```

CHECK -- Wait for and Test Completion of READ or WRITE Operation (R)

The CHECK macro instruction (for BSAM) waits, if necessary, for completion of an I/O operation requested by a READ or WRITE macro instruction and detects any errors and exceptional conditions that may occur. If read or write operations are completed normally, the program resumes execution at the instruction after the CHECK macro instruction.

As required, the CHECK macro instruction passes control to appropriate exits that are specified by the user in the data control block for error analysis (SYNAD) and end-of-data set (EODAD). The CHECK macro instruction automatically initiates volume switching for input data sets. Additional space for output data sets is automatically obtained when current space is filled and more WRITE macro instructions are issued.

The user must issue a CHECK macro instruction to test the I/O operation associated with a data event control block (DECB) before modifying or reusing it.

Name	Operation	Operand
[symbol]	CHECK	decb-{addrx / (1)}

decb
: specifies the data event control block (DECB), created as part of the expansion of a READ or WRITE macro instruction.

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

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: The CHECK macro instruction must be used to test for 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. A CHECK must be issued before

the number of outstanding READ or WRITE macro instructions exceeds the DCBNCP count (specified in the DCB macro instruction) in the data control block for the data set.

If the CHECK macro instruction tests a DECB that has not been posted as complete, the user's task waits until the event is completed.

If the CHECK macro instruction tests a READ operation that attempted to gain access to a block after the last block of a data set had been read, control is passed to end-of-data set exit (EODAD) whose address is provided in the EODAD field of the data control block. The task is abnormally terminated if an EODAD address is not supplied. Refer to Appendix C for contents of registers when the EODAD routine is entered.

If the CHECK macro instruction determines that the READ or WRITE operation was not completed correctly because of an I/O error, control is given to the user's Synchronous Error Exit (SYNAD) routine. Refer to Appendix B.

The RETURN macro instruction may be used to return to the calling program from the SYNAD routine. The program may then proceed, if desired, as if an error had not occurred. For input from any device, or for output to a unit record device, processing may be continued. In all other cases, the data control block should be closed.

The task is terminated if an error is detected by the CHECK macro instruction and the user has not provided a SYNAD routine.

If the CHECK macro instruction detects an end-of-volume condition, when processing a multivolume data set, processing continues with the next volume. If there are no additional volumes, the user's EODAD routine is entered.

A hardware-detected incorrect-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 length of the block actually read, the user can examine the channel status word (part of status indicators pointed to in the DECB) after issuing the CHECK macro instruction. The first byte of a format-U record read backwards from magnetic tape may be located by the same method.

The following table lists the results of incorrect-length error in which length of the record read is different from the DCBBLK for formats-F and U, or the LL field for format-V.

RECFM	Control Passed To SYNAD
Fixed (F)	Yes
Fixed blocked (FB)	If block is short by a nonmultiple of LRECL
Fixed standard (FS)	Yes
Fixed blocked standard (FBS)	If block is short by a nonmultiple of LRECL*
Variable (V)	Yes
Variable blocked (VB)	Yes
Undefined (U)	No

*If block is short by a multiple of LRECL, next record causes an end-of-volume condition. If current volume is last of the data set, control is passed to EODAD. If current volume is not the last, processing continues on next volume.

EXAMPLE: The CHECK macro instruction tests for completion of I/O operations in the order in which they are requested. The operand field contains the name of the data event control block specified in the read or write request.

```
            .
            .
            .
EX1     READ      INDECB,SF,INVEN,WORK,100
            .
            .
            .
        CHECK     INDECB
            .
            .
            .
EX2     WRITE     OUTDECB,SF,MNTHRPRT,WORK,100
            .
            .
            .
        CHECK     OUTDECB
            .
            .
            .
```

DQDECB -- Remove Unchecked DECBs From a Data Set's DECB Queue (R)

The DQDECB macro instruction (for BSAM) removes all unchecked DECBs from a queue of unchecked DECBs maintained by the system. If all of the DECBs within the queue have not been posted complete, the I/O requests associated with them are purged. DQDECB will not proceed until all DECBs have been posted complete either due to the purge or the fact that they have actually completed.

Name	Operation	Operand
[symbol]	DQDECB	decb-{addrx / (1)}

decb
specifies a data event control block (DECB) associated with the data set for which the DECB dequeueing will be performed. The DECB need not currently be in the DECB queue.

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

PROGRAMMING NOTES: The DQDECB macro instruction is normally used in the SYNAD routine when multiple READ or WRITE macro instructions have been issued without an intervening CHECK. If DQDECB is issued, all unchecked READ or WRITE requests must be reissued. The I/O operations associated with the data set that were not checked are removed from the system. If any of these DECBs are checked after the DQDECB without an intervening READ or WRITE, the CHECK will be treated as a NOP.

This facility is of use to users of the IMSK facilities of the DCB when they have multiple READ or WRITE requests unchecked and want to initiate their own error retry procedures, or to the user with multiple unchecked READ or WRITE requests who wants to reinitiate the sequence of I/O operations.

Upon return from DQDECB, register 0 contains a count of the number of unchecked DECBs in the queue, and register 1 contains a pointer to the list of unchecked DECBs. This queue is read-only and is only valid until the next I/O operation is initiated on the data set.

GETBUF -- Get a Buffer From a Pool (R)

The GETBUF macro instruction (for BSAM) obtains a buffer from a specified buffer pool. Buffers acquired by a GETBUF must be returned by a FREEBUF before they may be obtained again.

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 register 1 before execution of this macro instruction.

register
specifies a general register into which the control program is to place the address of the buffer.

CAUTION: The following error conditions result in termination of the task:

1. The dcb operand specifies an invalid data control block.
2. Buffer size is 0 or greater than 32,760.

3. Number of buffers in pool is 0 or greater than 255.
4. Data control block not open.

The address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: A buffer pool must have been assigned to the data control block by use of a GETPOOL, or the buffer option in the DCB macro instruction; i.e., BUFL= and BUFNO= are supplied in the DCB macro instruction. Each successive GETBUF macro instruction issued obtains a buffer in the order in which it exists in the buffer pool. For example, if a buffer pool contains five buffers, five successive GETBUF macro instructions would obtain five successive buffers from the buffer pool.

Buffers must be returned to the pool by the FREEBUF macro instruction before they can be obtained again.

If no buffer is available within the pool, the contents of the register specified in the GETBUF macro instruction will be set to zero rather than an address.

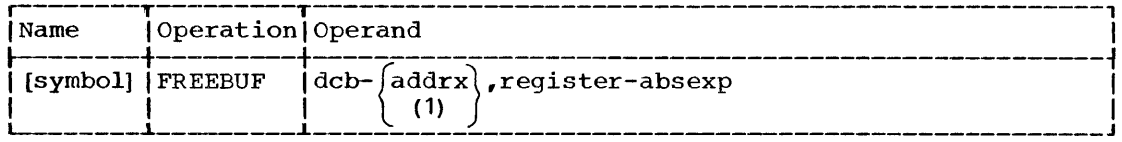
The address of the buffer pool is placed in the DCBBCN field of the data control block.

EXAMPLE: The GETPOOL macro instruction is used to define a buffer pool of 10 buffers of 100 bytes each. 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 data set is being read. (The length operand is not required in the READ macro instruction). The buffer is released by the use of the FREEBUF macro instruction.

```
GETPOOL                 INDCB,10,100
  .
  .
  .
OPEN                    (INDCB,(INPUT))
  .
  .
  .
GETBUF                  INDCB,(5)
  .
  .
  .
READ                    DECB1,SF,INDCB,(5)
  .
  .
  .
  .
  .
  .
FREEBUF                 INDCB,(5)
```

FREEBUF -- Return a Buffer to a Pool (R)

The FREEBUF macro instruction (for BSAM) returns a buffer (previously obtained by a GETBUF macro instruction) to a buffer pool so that it will be freed and can be obtained again by GETBUF. It is not necessary to free all buffers prior to issuing the CLOSE 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 being processed.

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 that contains the address of the buffer being returned to the pool.

CAUTION: Error conditions that result in termination of the task are:

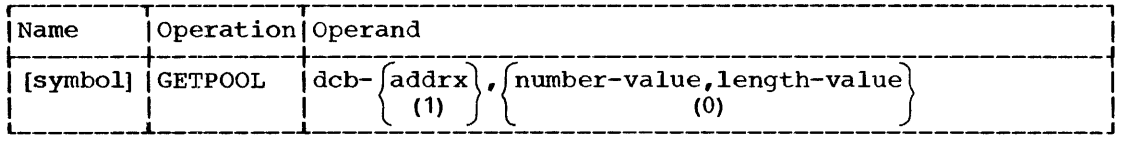
1. The dcb operand specifies the address of an invalid data control block.
2. Buffer pool address is not in data control block (GETBUF was not invoked before FREEBUF).
3. Buffer address specified by user does not belong to buffer pool.
4. Buffer specified by user is not in use (GETBUF was not used to obtain buffer).

The address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: To release a buffer by FREEBUF, a buffer pool must have been assigned to the data control block, and the specified buffer must have been obtained by the GETBUF macro instruction.

GETPOOL -- Get a Buffer Pool (R)

The GETPOOL macro instruction (for BSAM) requests allocation of an area of virtual storage for use as a buffer pool. 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 be in parameter register 1 prior to 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 is increased, if necessary, by the GETPOOL routine to be a doubleword

multiple. The maximum value is 32,760 bytes. If (0) is written, the value giving the number of buffers must be in the two high-order bytes of register 0, and the value giving the length of each buffer must be in the two low-order bytes of register 0, prior to execution of the macro instruction.

CAUTION: Failure to observe these restrictions results in termination of the task.

1. Only one buffer pool may be assigned to a data control block at one time.
2. Buffer length must be less than, or equal to, 32,760.
3. Number of buffers must be less than, or equal to, 255.

The address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: If the GETPOOL macro instruction is used, it must be executed prior to the execution of any GETBUF macro instruction which refers to the buffer pool area allocated by GETPOOL.

The FREEPOOL macro instruction should be issued to return the allocated buffer pool to the system, unless a CLOSE is issued for the data control block to which the buffer pool is assigned.

EXAMPLES: EX1 constructs a buffer pool consisting of two buffers, each 136 bytes long, in an area of virtual storage. This buffer pool is assigned to the data control block REPORT. EX2 indicates that the required parameters were in registers 0 and 1 prior to execution of the macro instruction.

```
EX1     GETPOOL   REPORT,2,136
EX2     GETPOOL   (1),(0)
```

FREEPOOL -- Free a Buffer Pool (R)

The FREEPOOL macro instruction (for BSAM) releases an area that had previously been assigned as a buffer pool to a specified data control block. The area must have been acquired through either the execution of a GETPOOL macro instruction or by the buffer option described in the DCB macro instruction; i.e., when the DCB macro instruction was written, BUFNO= and BUFL= were included.

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 be in parameter register 1 prior to execution of this macro instruction.

CAUTION: If the dcb operand does not specify the address of a valid data control block, the task is terminated.

The address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: If the associated data set is processed by means of BSAM, FREEPOOL may be issued as soon as the buffers are no longer required.

The area released by FREEPOOL must have been acquired through the execution of a GETPOOL macro instruction, or by the buffer option described in the DCB macro instruction; i.e., BUFNO= and BUFL= were supplied in the data control block.

EXAMPLES: EX1 releases the buffer area assigned to the data control block OUTPUT. EX2 releases the buffer area assigned to the data control block whose address is in register 1.

```
EX1     FREEPOOL  OUTPUT
EX2     FREEPOOL  (1)
```

BSP -- Backspace a Block (R)

The BSP macro instruction (for BSAM) backspaces a block on the current magnetic tape or direct-access volume. Backspacing is always toward load point (or its equivalent on direct-access) regardless of the OPEN macro instruction's parameters or the direction of reading.

This macro instruction is applicable only to magnetic tape or a direct-acces device and becomes a NOP for other devices.

Name	Operation	Operand
[symbol]	BSP	dcb-{addrx / (1)}

dcb
specifies address of data control block opened for the data set to be backspaced.

If (1) is written, the data control block address must have been loaded into parameter register 1 before executing this macro instruction.

CAUTION: Abnormal termination occurs if:

1. Data control block specified by the user is not validly opened.
2. Track overflow option is specified.
3. All read and write operations have not been checked for completion.

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: Following execution of the BSP macro instruction, register 15 contains a return code of 0 if the operation is completed normally. It also contains a return code of 0 if the operation encountered a permanent positioning error, in which case the next CHECK of a READ or WRITE passes control to the SYNAD routine.

If a tape mark is encountered on a backspace, the tape is repositioned to the tape position it was at before BSP was issued and a return code of 4 is placed in register 15.

If user attempts to backspace into a header or trailer label track on direct-access volumes, backspacing does not occur and a return code of 4 is placed in register 15.

All read or write operations must be checked for completion before the BSP macro instruction is executed.

If more than one BSP is issued without an intervening READ or WRITE, then NOTE, POINT, or CNTRL macro instructions may be more efficient.

CNTRL -- Control On-Line Input/Output Devices (R)

The CNTRL macro instruction (for BSAM) performs non-data-transfer operations on magnetic-tape drives and online card readers and printers. The functions provided are: card-stacker selection, printer carriage control, and magnetic-tape repositioning.

Since online card readers and printers cannot be addressed directly by most users of the system, only users with proper system authorization may use this macro instruction for card-stacker selection and printer carriage control. All users, however, may use this macro instruction for magnetic-tape repositioning.

Name	Operation	Operand
[symbol]	CNTRL	dcb-{addrx | (1)},{action-code [,number-value] | (0)}

dcb
: specifies the address of data control block opened for data set being processed. If (1) is written, the address must be loaded into parameter register 1 before execution of this macro instruction.

action
: specifies, by a code, the service to be performed:

  SS - selects a stacker for a card reader (stacker 1 or 2).

  SP - spaces lines on a printer; space = 1, 2, or 3.

  SK - skips to channels 1 through 12 on carriage control tape for a printer.

  BSR - backspaces over a specified number of blocks on magnetic tape. One block is assumed if number operand is omitted. BR is the abbreviated code.

  BSM - moves backward past a tape mark and forward spaces over the tape mark. A number value of 1 is always assumed. BM is the abbreviated code.

  FSR - forward spaces over a specified number of blocks on magnetic tape. One block is assumed if number operand is omitted. FR is the abbreviated code.

  FSM - moves forward past a tape mark and backspaces over the tape mark. A number value of 1 is always assumed. FM is the abbreviated code.

FSF - moves forward past a tape mark. A number value of 1 is always assumed. FF is the abbreviated code.

BSF - moves backward past a tape mark. A number value of 1 is always assumed. BF is the abbreviated code.

WTM - writes a tape mark on magnetic tape. A number value of 1 is always assumed. WM is the abbreviated code.

REW - rewinds magnetic tape. RW is the abbreviated code.

RUN - rewinds and unloads magnetic tape. RU is the abbreviated code.

ERG - executes an erase gap for magnetic tape. ER is the abbreviated code.

If (0) is written, the two-character action code must be placed in the two high-order bytes of parameter register 0 before execution of this macro instruction. In the case of three-character action codes, the abbreviated code must be placed in those bytes.

number
specifies a value for the stacker number, number of lines to be skipped on the printer, printer carriage-tape channel, or number of blocks on magnetic tape to qualify the action operand. Maximum value is 32,767. If (0) is written, the value must be placed in the two low-order bytes of parameter register 0; value is a binary integer.

CAUTION: If magnetic-tape positioning is performed, an uncorrectable tape-spacing error results in linkage to the user's SYNAD routine; this does not apply to action codes SS, SP, SK, REW, or RUN. Refer to Appendix B for a discussion of SYNAD.

Abnormal termination occurs if:

1. Action code is undefined or not applicable.
2. Number parameter is undefined for the action parameter.
3. A SYNAD-type error occurs and the user has not provided a SYNAD address.
4. Data control block specified by the user is not a validly opened data control block.
5. Outstanding read or write operations have not been checked.

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: For stacker selection, the DCBNCP field of data control block must be 1. Each READ macro instruction directed to a card reader must be followed by a CHECK macro instruction and a stacker selection CNTRL macro instruction directed to the same device. Stacker selection is not available for card punch through the use of the CNTRL macro instruction; however, the user may specify the desired stacker selection by changing, in his program, the DCBSTA field in the data control block. See Appendix F.

READ and WRITE operations must be checked for completion before the CNTRL macro instruction is issued. If used to control stacker selec-

tion, the CNTRL macro instruction must be issued for each read operation.

For printers, a skip to a specified channel results in no action if the device is already at that channel.

Control is returned to user if a tape mark or a load point is encountered during an attempt to forward space or backspace blocks; control is not passed to the SYNAD routine. Register 15 contains binary zeros if operation is completed normally; otherwise, it contains a count of the remaining number of forward spaces or backspaces that were not completed in its low-order two bytes.

FEOV -- Force End of Volume (R)

The FEOV macro instruction (for BSAM) directs TSS/360 to advance to the next volume of a data set before the end of the current volume is reached. This macro instruction is applicable only to data sets mounted on magnetic tape or direct-access devices.

Name	Operation	Operand
[symbol]	FEOV	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.

CAUTION: The following errors cause the results indicated:

Errors	Result
The dcb operand specifies address of a data control block that is not open	No action
The dcb operand specifies address of an invalid data control block	Task terminated
Data set is not processed by BSAM (magnetic tape or direct-access devices)	Task terminated
Not all BSAM READs and WRITEs to data set are checked.	Task terminated

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

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
```

POINT -- Position to a Block (R)

The POINT macro instruction (for BSAM) repositions a magnetic-tape or direct-access volume to a specified block within a data set on that volume. Thus the POINT macro instruction permits reading or writing of a sequential data set from any specified position.

The NOTE macro instruction may be used to provide the positioning information that is required for the POINT macro instruction.

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 fullword boundary.

  If (0) is written, the address of the block identification must be loaded into parameter register 0 before execution of this macro instruction. The initial relative address for the first record on a direct-access device is (TT=0,R=0). The initial relative address for the first record on a magnetic tape device which was not opened for RDBACK or MOD is (CC=0). The initial relative address for the first record on a magnetic tape which was opened for RDBACK or MOD is CC=(Block Count from trailer label).

CAUTION: Abnormal termination occurs if the data control block specified by the user is not validly opened.

Executing a POINT macro instruction for a direct-access device 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, upon execution of a CHECK macro instruction, causes control to be given to the user's SYNAD routine. If an error occurs during the positioning of magnetic tape, the POINT macro instruction passes control immediately to the SYNAD routine.

The POINT macro instruction must not be issued for a data set on an unlabeled magnetic tape volume or one containing nonstandard labels, if the data set is opened under either of these conditions:

1. DDEF macro instruction or command specifying disposition parameter of MOD.
2. OPEN macro instruction specifying RDBACK.

The POINT macro instruction is applicable only to direct-access and magnetic-tape devices. An immediate return with no action is taken for other devices.

For direct-access volumes, a user may reposition to any point on the volume that is assigned to the data set. For a magnetic-tape volume which is opened for OUTPUT, OUTIN, or INOUT, the user must not reposition beyond the last record written.

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: 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.

If the POINT macro instruction is used on a data set opened in the UPDAT mode, a READ macro instruction must be issued following the POINT macro instruction. The WRITE (update mode) specifies that only the last record read may be updated and that the updated record written must replace the last record read. Since a POINT macro instruction is used to alter the next sequential I/O address, a WRITE (update mode) does not return the updated record to the correct address.

If the user points to a block with a count provided by a NOTE issued after a read-backward, another read backward should be executed to read the block provided by the previous NOTE. A read-forward does not read the block specified by the NOTE.

A POINT macro instruction executed after a WRITE macro instruction returns the identification field of the block just written. To reposition so that writing will begin at the next block, the user should add binary 1 to the low-order byte of the field. For a direct-access device, a binary 1 is added to the Z-byte of the TTRZ field. For magnetic tape, the binary 1 is added to the low-order C of the ZZCC field.

EXAMPLE: In the following example, the POINT macro instruction is used to reposition a volume to a block which was identified previously by a NOTE macro instruction.

```
    .
    .
    .
WRITE     OUTDECB,SF,MYDCB,(4),100    This is the record to which
                                      the program will reposition.
    .
    .
    .
CHECK     OUTDECB
FREEBUF   MYDCB,4
    .
    .
    .
NOTE      MYDCB                       Note the position of the rec-
ST        1,SAVE                      ord under consideration.
    .
    .
    .
GETBUF    MYDCB,4                     Reposition to the record
POINT     MYDCB,SAVE                  being considered and
READ      INDECB,SF,MYDCB,(4),100     read it.
    .
    .
    .
```

NOTE -- Provide Position Feedback (R)

The NOTE macro instruction (for BSAM) causes the relative position within a volume of a block just read or written to be placed in register 1. This relative position identifies the block for subsequent repositioning of the volume.

NOTE provides a block count for magnetic tape. For direct-access volumes, the count is the track number relative to the beginning of the data set portion on the volume and the record number within the track.

The NOTE macro instruction normally provides information for a subsequent POINT macro instruction.

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 data control block address must have been loaded into parameter register 1 before execution of this macro instruction.

CAUTION: Abnormal termination occurs if the data control block specified by the user is not validly opened.

For a data set on magnetic tape, the NOTE macro instruction should not be issued for an unlabeled data set or a data set containing nonstandard labels, if the data set is opened under either of these conditions:

1. DDEF macro instruction or command has a disposition parameter of MOD.
2. OPEN macro instruction specifies RDBACK.

The current block count in the data control block is not valid under the above conditions.

For a data set on magnetic tape, a NOTE macro instruction issued after a POINT macro instruction, without an intervening READ or WRITE macro instruction, does not return the relative address of the last record read or written. NOTE returns the data control block count minus 1, if the last I/O operation was not a READ (type SB); or it returns the data control block count plus 1, if the last I/O operation was a READ (type SB).

The address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: All READ or WRITE requests must be checked for completion before the NOTE macro instruction is executed. The provided block identification is always within the current volume.

Following execution of the NOTE macro instruction, the system places the block identification of the last block read or written in parameter register 1.

The form of the block identification depends on whether magnetic tape or direct-access devices are being used as follows:

Magnetic Tape: If magnetic tape is used, the block identification is a four-byte block count of the form zzCC, where:

zz - binary zero bytes;
CC - the block number (binary) within the volume.

The block identification may be used in the POINT macro instruction to reposition the magnetic tape to the location of the block.

Direct-Access Device: If a direct-access device is used, the block identification is a four-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 0).
R - the block number on that track (first data block equals 0).
z - a binary zero byte.

If the last operation was a WRITE, an additional parameter is provided by NOTE in register 0 in the form zzLL, where:

zz = binary zero bytes.
LL = the number (in binary) of bytes remaining on that track.

The initial relative address for the first record on a direct-access device is (TT=0, R=0). The initial block count for the first record on a magnetic-tape device which was not opened for RDBACK or MOD is (CC=0). The initial block count for the first record on a magnetic tape, which was opened for RDBACK or MOD, is CC minus 1 (CC=Trailer Label Block Count). NOTE is applicable only to direct-access and magnetic-tape devices. The address, sent back in general register 1 for any other equipment type, is the data control block count minus one and is preceded by two bytes of binary 0.

PRTOV -- Test for Printer Carriage Overflow (R)

The PRTOV macro instruction (for BSAM) controls the page format for an on-line printer. The user may test channel 9 or 12 of the printer control tape to determine if an overflow condition exists. Since the printer cannot be addressed directly by most users of the system, this macro instruction may be issued only by users with proper system authorization.

Before testing overflow indicators, PRTOV waits for completion of all previously requested printing.

Name	Operation	Operand
[symbol]	PRTOV	dcb-{addrx | (1)},number-{9 | 12}[,userrtn-{addrx | (0)}]

dcb
specifies the address of the data control block opened for the data set being processed.

If (1) is written, the data control block address must be loaded into parameter register 1 before execution of this macro instruction.

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 the appropriate program indicator (for channels 9 or 12) is on when tested. If this operand is omitted, and if the overflow condition exists, an automatic skip to channel 1 is performed prior to the next WRITE operation.

CAUTION: Abnormal termination occurs if the data control block specified by the user is not validly opened.

The address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: This macro instruction is applicable only for a printer.

The user routine, if it uses a PSECT, must use the same PSECT as the routine that issues the PRTOV macro instruction. To continue processing at the point at which PRTOV macro instruction was issued, the user routine must branch to the address that was contained in general register 14 upon entry to the user routine. A RETURN macro instruction may not be used for this purpose.

If no user routine is specified, execution of the problem program continues after a PRTOV macro instruction is issued. When the line associated with the first WRITE macro instruction issued after the PRTOV is to be printed, the appropriate program indicator is tested. An automatic skip to channel 1 is performed if an overflow has occurred.

If a user routine is specified, the control program waits after a PRTOV macro instruction is issued. When all prior print operations are complete, the appropriate program indicator is tested.

Upon entry to the user's overflow routine, the contents of the general registers are:

Register	Contents
0	Unspecified
1	Address of data control block
2 to 13	Same as existed before macro instruction was executed
14	Return address
15	Address of userrtn routine

EXAMPLES:

```
EX1     PRTOV     OUTDCB,9
EX2     PRTOV     PRINTDCB,12,OVERFLOW
```

In EX1, an overflow condition on channel 9 of the printer-control tape results in an automatic skip to channel 1 since the operand, userrtn, is omitted. In EX2, an overflow condition on channel 12 results in control passing to the user's overflow routine.

QUEUED SEQUENTIAL ACCESS METHOD

The queued sequential access method (QSAM) consists of the TSS/360 data set management facilities that enable a user to access blocked or unblocked physical sequential data sets. QSAM, in contrast to BSAM, permits the programmer to store and retrieve records of a sequential data set without coding his own blocking/deblocking and buffering routines. A sequential data set can be stored on, or retrieved from, disk, tape, or cards, and can be printed out by a printer. The record format within each such data set can be fixed length (blocked or unblocked), variable length (blocked or unblocked), or undefined length (unblocked only). Such attributes are unique for each data set; they must be defined to the system before a data set can be accessed by QSAM. The macro instructions provided to a user, by QSAM, for accessing a data set in an appropriate manner, are summarized below.

GET used for reading logical record in a sequential order. The initial GET reads in a physical record transferring it from the input device to a system maintained buffer area and, when the physical record is blocked, locates the first sequential logical record within the physical record. Each subsequent GET locates the next sequential logical record within the physical record until all logical records within that physical record have been processed; then the system reads in another physical record automatically and locates logical records as indicated above.

PUT for writing new or altered logical records into a physical sequential output data set.

PUTX for writing an updated or identical logical record directly from an input data set to an output data set, without altering the length of the record. The next sequential logical record contained in an input buffer area (where it may have been modified) is transferred to the output buffer as the next sequential output record. The system must be positioned at the next sequential logical input record by issuing a locate mode GET macro instruction prior to the PUTX

RELSE causes the remaining records of the current input buffer to be ignored, locates the next sequential physical record's input buffer area and positions the user at the first logical record in that buffer area. The next GET macro instruction will retrieve the first logical record from the new input buffer.

TRUNC causes the current output buffer to be regarded as filled, transfers a physical record from that output buffer to the output device, and positions the system at the next buffer area. The next PUT issued causes the user to be positioned at the new output buffer area in which he can construct the next logical record.

SETL enables a user to logically position a data set at its beginning, end, at the previous record, or at any logical record within a blocked sequential data set. Subsequent PUT or GET operations will start at this new position.

CNTRL provides control for card stacker selection, printer carriage control, and magnetic tape positioning

PRTOV controls the page format for an on line printer by testing channels 9 and 12 on the printer control tape, as overflow indicators, and allowing the user to provide an overflow routine to reposition the printer at any desired channel of the printer control tape.

Detailed explanations of each of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to QSAM data set management and the related macro instructions can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

GET -- Get a Logical Record (R)

The GET macro instruction (for QSAM) can be specified in either locate mode or move mode. In locate mode, the GET macro instruction locates the next sequential logical record in an input data set and places its address in register 1. The user may then operate on the record where it is or move it to a work area. The logical record pointed to by register 1 resides in an input buffer where a system-scheduled read operation placed it. In move mode, the GET macro instruction acquires the next sequential logical record and automatically moves it from the input buffer to an area in virtual storage specified by the user.

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 the macro instruction.

area (for move mode only)
specifies the address of the user's work area into which the record is to be moved. If (0) is written, the address must have been loaded into parameter register 0 before execution of the macro instruction.

CAUTION: If either of the following error conditions exists as a result of the execution of the GET macro instruction, control will be passed to the Synchronous Error Exit (SYNAD) routine specified in the data control block;

1. The next record to be processed starts a block that could not be read satisfactorily because of an error condition.

2. A preceding PUTX macro instruction could not be executed without resulting in an error condition. This situation is discovered by the GET macro instruction when working in update mode.

3. When processing variable length records, the length of a block (LL) does not equal the actual block size.

4. When processing variable length records, the lengths of each individual record (ll) within a variable length block do not add up to the length indication of the block (LL).

When the SYNAD routine is given control, the general registers and status indicators are set as shown in Appendix B.

The address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: In locate mode, the control program returns the address of the next logical record in parameter register 1, and places the record length in the logical record length (DCBLRECL) field of the data-control block. In the move mode, the area address provided by the user is returned in register 1 and the logical record length of the accessed record is placed in DCBLRECL. Because QSAM does not support the substitute-mode GET macro instruction, this feature (i.e., return of the area address) provides compatibility which allows the time sharing system to use the move mode in order to execute programs originally written to use the substitute-mode GET.

If a GET is requested beyond the end of a data set, as a result of sequential operation or the user EODAD exit is taken, see Appendix C.

EXAMPLE: In the following example written in the move mode the next record from the data set associated with the DCB labeled STAT is moved to the workarea labeled SAMPLES. The address of the word area is returned to the user in parameter register 1.

```
EX1       GET   STAT,SAMPLES
                .
                .
                .
STAT      DCB   DSORG = PS, ....
                .
SAMPLES   DS    20F
                .
```

PUT -- Include a Record in an Output Data Set (R)

The PUT macro instruction (for QSAM) may be specified in either locate mode or move mode. In locate mode, the PUT macro instruction places in register 1 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 record for incorporation into the output data set. In move mode the PUT macro instruction moves a record from a user specified area in virtual storage to an output buffer. When an output buffer is filled, the system places its contents into the output data set.

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. In locate mode, after execution of the macro instruction the address of the next buffer segment large enough to hold the next logical record is returned in register 1.

area (for move mode only)
specifies the address of the 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.

CAUTION: Any exceptional condition resulting from the execution of a PUT macro instruction causes control to be passed to the user's synchronous error exit (SYNAD) routine. In this case, the general register and the exceptional condition fields in the data control block are set as shown in Appendixes B and F.

The address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: Before executing this macro instruction, the user must place the length of the record in the logical record length field (DCBLRECL) of the data control block according to the format of the logical records as follows:

For format-F records, the logical record length is taken from DCBLRECL. This field should not be altered after the DCB is opened or an incorrect length block will be written. This will cause abnormal termination.

For format-U records, the actual record length must be known before the record is constructed, and must be placed in the DCBLRECL field. Abnormal termination will occur if DCBLRECL is greater than DCBBLKSI.

For format-V records, one of the following procedures must be chosen depending upon whether locate mode or move mode is used. For locate mode the actual record length must be placed in the DCBLRECL field or anestimated record length (not less than the actual record length) must be placed in the DCBLRECL field. If the estimated record length in DCBLRECL is greater than DCBBLKSI, an abnormal termination will occur. For move mode, the length '11' of each logical record determines the amount of buffer space needed. If '11' is greater than DCBBLKSI, an abnormal termination is taken. For PUT move mode, the area address provided by the user is returned in register 1. Because QSAM does not support the substitute mode PUT macro instruction, this feature (return of the area address) provides the compatibility which allows move mode to be used in order to execute programs originally written for OS/360 which use substitute mode PUT.

EXAMPLE: In the following example, the use of a move-mode PUT macro instruction 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 is processed 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
```

PUTX -- Include a Logical Record in an Output or Updated Data Set (R)

The PUTX macro instruction (for QSAM) causes the next logical record contained in a buffer area of an input data set to be written as the

next sequential logical record of an output or updated data set. This macro instruction may be specified in either update mode or output mode. In update mode only, the output and input data sets are one and the same, and only the dcbout-addrx operand is required. In output mode, two different data sets are used, necessitating that both operands must be specified.

Name	Operation	Operand
[symbol]	PUTX	dcbout-{addrx / (1)}[,dcbin-{addrx / (0)}]

dcbout-addrx
: specifies the address of the data control block opened for the output data set. In the update mode the output and input data sets are the same and only the dcbout-addrx operand is required. If (1) is written, the address must have been loaded into parameter register 1 before execution of this macro instruction. The DCB referred to in the dcbout operand must be opened for UPDAT if the update mode is used or it must be opened for OUTPUT if the output mode is used.

dcbin-addrx
: 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:

- The data set must reside on a direct access device.
- 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 in that block have been processed.
- The length of the block and the length of each logical record cannot be altered.
- Additional logical records cannot be inserted into the block nor can existing logical records be deleted from the block.

PROGRAMMING NOTES: Any exceptional condition resulting from the execution of a PUTX macro instruction causes control to be passed to the user's synchronous error exit (SYNAD) routine.

The PUTX macro instruction must always be preceded by a locate mode GET macro instruction. This GET macro instruction must specify the same data set as specified by an update mode PUTX macro instruction, or it must specify the data set that is used as input by an output mode PUTX macro instruction.

Since the update mode uses only a single data set the user need only issue a PUTX for those logical records which are to be updated. Those records which have not changed can be bypassed, and thereby remain unchanged, simply by issuing two successive GET macro instructions (see EXAMPLE below.)

In output mode two distinct data sets are used and a PUTX is required for each logical record that is to be included in the output data set being created. Abnormal termination will occur if these requirements are violated.

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 2 indicates which combinations of input and output record formats are acceptable.

Table 2. Acceptable record formats 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	S	---	---	---	---
from F	S	S	S	---	---
from FB	S	S	S	---	---
from V	S	---	---	S	S
from VB	S	---	---	S	S

where:

---	indicates unacceptable record format combination
S	indicates acceptable record format combinations (only simple buffering supported by TSS)
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 2:

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 set.
2. The logical record size for format-F and -FB records must be the same for both data sets.
3. The maximum logical record for format-V and -VB records must correspond.

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 DCBA
     .
     .
     .
     B LLS
```

RELSE -- Release an Input Buffer (R)

The RELSE macro instruction (for QSAM) 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 block.

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 DCB 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.

If a RELSE is issued before the first GET of the data set, the macro instruction is ignored.

PROGRAMMING NOTES: If a data set is being read backwards, the RELSE causes the same results as in forward reading.

TRUNC -- Truncate an Output Buffer (R)

The TRUNC macro instruction (for QSAM) causes the current output buffer to be regarded as filled. The next PUT macro instruction will use the next block 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 DCB 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.

PROGRAMMING NOTES: Any exceptional condition resulting from the execution of a TRUNC macro instruction causes control to be passed to the user's synchronous error exit (SYNAD) routine.

If a TRUNC is issued on a data set OPEN'ed for UPDAT, the following GET will retrieve the first logical record from the next block. The last block will be written out including all logical records read plus those not updated by a PUTX.

If a TRUNC is issued before the first PUT of a data set, the TRUNC macro instruction is ignored.

CNTRL -- Control a Printer or Stacker (R)

The CNTRL macro instruction (for QSAM) provides stacker selection of an on-line card reader, or carriage control of an on-line printer.

Name	Operation	Operand
[symbol]	CNTRL	dcb-{addrx | (1)},{action-[SS|SP|SK],[number-value] | (0)}

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 position.

CAUTIONS: If stacker selection is desired and unblocked records are being read. Each GET macro instruction must be followed by a stacker-selection CNTRL macro instruction directed to the same device. The CNTRL macro instruction need not immediately follow the GET macro instruction. GET (locate mode) or GET (move mode) must be used exclusively for a card reader.

If stacker selection is desired and blocked records are being read, CNTRL should be issued only after the GET which refers to the last record in the block.

CNTRL need not be issued for the GET which invokes EODAD.

For the printer, use of control characters does preclude use of the CNTRL macro instruction.

If a locate mode PUT was last issued before the CNTRL, the SP or SK CNTRL function will occur immediately following the line associated with the PUT preceding the locate mode PUT. If a move mode PUT was last issued before the CNTRL, the SP or SK CNTRL function will occur immediately following the line associated with the move mode PUT.

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 (for QSAM) 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 | (1)},number-{9|12}[,userrtn-{addrx | (0)}]

dcb
: specifies the address of the data control block opened for the dataset 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.

PROGRAMMING NOTES:

Existence of an overflow condition, as indicated by the channel 9 or 12 machine indicator, is detected by the system and retained in corresponding program indicators, one for each channel. The control program resets the appropriate program indicator only when a PRTOV macro instruction tests that indicator. Thus, the PRTOV macro instruction detects an overflow condition that occurred in any prior, completed operation that was not tested. Testing occurs as follows:

If no user routine is specified, execution of the problem program continues after a PRTOV macro instruction is issued. When the line associated with the first PUT macro instruction issued after the PRTOV is about to be printed, the appropriate program indicator is tested.

If a user routine is specified and a move mode PUT preceded the PRTOV, the control program WAITs after a PRTOV macro instruction is issued. When all prior PUT operations are complete, the appropriate program indicator is tested.

If a user routine is specified and a locate mode PUT was last used, the overflow indicator will be tested to indicate the status of the print line associated with the PUT (locate or move mode) which preceded the locate mode PUT. A locate mode PUT does not cause a line to be printed until the next PUT or TRUNC.

This macro instruction causes no action, if used for a device other than a printer. The USERRTN must have the same PSECT as the routine which issued the PRTOV. To continue processing at the point at which the PRTOV macro instruction was issued, the USERRTN must branch to the address which was contained in general register 14 upon entry to USERRTN, and must not issue a RETURN macro instruction.

The contents of the general registers upon entry to the user's overflow routine are as follows:

Register	Contents
0	Unspecified
1	Address of the data control block (DCB)
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
```

SETL -- Specifies Start of Sequential Processing (R)

The SETL macro instruction (for QSAM) enables the user to position himself at the beginning, end, previous record, or at any logical record within a sequential data set volume.

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 DCB 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
C	After this instruction is executed DCBLPDQ will contain the current retrieval address for use by a SETL type code R instruction.
R	Retrieval address specified in the llimit parameter as obtained from DCBLPDQ in the data control block, following a SETL type code-C.
B	Beginning of data on current volume.
E	End of data on current volume. On OUTPUT data sets this is the current position.
P	Previous logical record in volume (backspace).

llimit
specifies the address of a field containing the retrieval address which must be a double word oriented, 6-byte field.

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

Only if the type code specifies R should the llimit field be provided.

CAUTION: A SETL issued for a data set or opened for UPDAT must be followed by a GET locate mode macro instruction before a PUTX can be issued.

If a SETL with type E code is given for magnetic tape, subsequent use of a SETL with type C or R codes will be invalid.

If SETL is used, the user must specify the SETL options in the MACRF field of the data control block.

If type contains a P, a SETL issued for a direct access volume with track overflow specified in the DCB causes no action to be taken.

If type contains R, a SETL cannot be issued for an unlabeled magnetic tape volume which was OPEN'ed for RDBACK or if MOD was specified.

The retrieval address obtained from the DCBLPDQ field cannot be altered before it is furnished to the SETL routine in the llimit parameter. SETL type code C must be issued just before the retrieval address in DCBLPDQ is saved for use by a subsequent SETL type code R.

The execution of a SETL macro instruction on a direct access device results in an error if a volume cannot be properly repositioned or if the DCBLPDQ is invalid. These errors cause the SETL to pass control to SYNAD.

If repositioning errors occur in the execution of a SETL on a magnetic tape, control passes immediately to SYNAD.

PROGRAMMING NOTES: The DCBLPDQ field is six bytes in length and provides the relative address in the volume of the last logical record processed by QSAM. The DCBLPDQ should not be altered by the user and is used when R is specified in the type-code operand. The end of the data set OPEN'ed for OUTPUT is the current address. If E is specified in the type-code operand for an output data set, a SETL will position the user to his current address. If R is specified, the limit parameter (DCBLPDQ saved) cannot exceed the address of the last PUT.

EXAMPLE:

```
OPEN          INDCB,INPUT
.
.
.
GET           INDCB              (1st logical record)
GET           INDCB              (2nd logical record)

Save DCBLPDQ in RETAIN

GET           INDCB              (3rd logical record)
.                                .
.                                .
.                                .
GET           INDCB              (nth logical record)
SETL          INDCB, R, RETAIN

GET           INDCB              (2nd logical record)
.                                .
.                                .
.                                .
GET           INDCB              (last record of volume)
.
.
.
CLOSE         INDCB
```

In the above example, the first GET after the SETL macro instruction will furnish the 2nd logical record. If B had been specified in the type-code operand, the 1st logical record would have positioned the user to the address of the logical record just beyond the last record of this data set stored on the volume. The next GET would have caused EODAD to be given control if current volume is the last in the data set. If not the last volume, the first record of the next volume is provided. If type contained a P, the nth logical record (previous logical record) would have been furnished by the next GET. (If E, B, C, or P is specified in the type operand, the llimit parameter is ignored).

INPUT OUTPUT REQUEST FACILITY

The input/output request facility (IOREQ) consists of the TSS/360 data set management facilities provided for users who would rather program their own I/O device control routines than employ those from the VAM or SAM access methods. It provides a means to control I/O devices through user specification of channel command words (CCWs) that are normally created by the TSS/360 supplied access methods. Using IOREQ, the user can create a series of these channel instructions and execute them as he desires. The TSS/360 macro instructions IOREQ, CHECK, and VCCW, have been provided to users who desire to, in effect, create their own specialized access methods.

As with TSS/360 access methods, before the IOREQ facilities can be used to access a data set, the data set must be described and connected to the system by previous user (or system) issuance of the DCB, DDEF, and OPEN macro instructions and/or DDEF command, and, when he has finished accessing the data set, he must disconnect the data set from the system via a CLOSE macro instruction.

IOREQ causes a request for the input/output operations specified by a user coded VCCW or a string of VCCW macro instructions to be recorded in a control block (DECB) and placed on an input/output request queue. Control is then returned to the user's program; the request is subsequently executed by the system when the device is available.

CHECK checks the queue of control blocks (DECBs), containing the requests for one, or many, input/output operations, to determine if these requests have been satisfied; if completed satisfactorily, control is returned to the next sequential instruction following the check macro instruction. It also indicates whether errors or exceptional conditions have occurred while attempting to satisfy the request.

VCCW generates a double word channel command word (i.e., CCW) containing all the information needed by the channel to execute the requested input/output activity. The desired I/O activity can then be initialized by the IOREQ macro instruction.

A detailed explanation of the above macro instructions and the format in which they may be specified are shown below. Further information pertaining to the input/output request facility and user handling of I/O operations can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C-28-2032.

IOREQ -- Request an Input/Output Operation (S)

The IOREQ macro instruction (for the IOREQ facility) initiates an input/output operation which is specified by a virtual channel command word (VCCW). See the VCCW macro instruction in this section.

After an IOREQ macro instruction is issued, control returns to the problem program before the I/O operation is completed. The CHECK macro instruction must be used to ensure the completion of the I/O operation.

Name	Operation	Operand
[symbol]	IOREQ	decb-symbol,type-{N|B},dcb-addr,vccw-addr, length-value,sio-value

decb
specifies the name to be assigned to the data event control block (DECB) built by the macro expansion.

type
specifies either:

N nonbuffered I/O operation
B buffered I/O operation

dcb
specifies the address of the data control block opened for this IOREQ.

vccw
specifies the address of a list of virtual channel command words built by the VCCW macro instruction.

length
specifies the number of VCCWs in the VCCW list to be issued.

sio
specifies the number of the VCCW in the list which is to be executed first.

PROGRAMMING NOTES: The IOREQ macro instruction builds a data event control block (DECB) which is addressed by the symbol coded for the decb operand.

The format of the DECB is:

Offset from DECB-symbol	Size in Bytes	Field
+0	1	Event Control Block (ECB)
+1	3	Reserved by the system (user must not alter)
+4	2	Type field (buffered or nonbuffered IOREQ)
+6	2	Length field (for buffered only)
+8	4	DCB address
+12	4	Data area address (for buffered only)
+16	4	Pointer to status indicators
+20	4	VCCW list address
+21	2	Used by the system (user must not alter)
+26	1	Sense byte 0
+27	1	Sense byte 1
+28	1	VCCW list length in doublewords
+29	1	Offset from VCCW list in doublewords to start VCCW
+30	2	Reserved by the system (user must not alter)
+32	8	Modified channel status word[1] (CSW)
+40	8	Sense bytes (1-8)

[1]Modified CSW differs only from CSW in that the first word contains the 32-bit address of the instruction causing unit check or unit exception.

The DECB used for IOREQ must not be altered until the operation has been checked.

If buffering is specified, the buffer built for read request VCCWs may have overlapping data areas. However, the complete buffer area needed for all the read request VCCWs must form a contiguous area. For write request VCCWs, unique buffer space is allocated for each VCCW regardless of whether the areas used by the VCCWs have overlapping por-

tions. Consequently, write request VCCWs do not have to form contiguous areas.

For buffered VCCW write requests: the contents of the given data address are used, when the IOREQ macro instruction is issued even if these contents will be changed by a read request in the VCCW.

Each IOREQ macro instruction, which causes an input/output request to be executed, accomplishes this request by building an IORCB. IORCBs are executed separately by the system unless they are "chained." Chaining IORCBs saves time if a following IORCB arrives in the system before the previous IORCBs commands are completed.

If chaining to the next IORCB is desired, the last instruction to be executed must be the last in the user's VCCW list and must have the IOC flag set. (This instruction is usually a NOP.) Chaining of IORCBs is accomplished by changing the last CCW in a command list to a TIC to the start command in the next IORCB. This start CCW cannot be a TIC, and must be executable only once. IORCB chaining is allowed only between IORCBs on the same device. When chaining is requested, it is still necessary to check each IOREQ result by using the CHECK macro instruction. When execution of the IOREQ macro instruction is completed, register 15 contains a return code in its low-order byte.

Return Code (decimal)	Significance
0	I/O initiated
4	The NCP value in the data control block is exceeded; (I/O not initiated) or DECB "active", or DECB in "wait" state.
8	I/O not initiated. The VCCW list contains an error. One of the first eight rules for forming VCCW lists has been violated (refer to the IOREQ: VCCW macro instruction).
12	I/O not initiated. The area needed for IOREQ is too large. Reduce or change VCCW list.

L- AND E-FORM USE: The L-form macro instruction results in a macro expansion consisting of only a parameter list. The E-form results in a macro expansion only consisting of executable instructions. The E-form macro instruction uses the DECB built for it by the L-form macro instruction.

If the E-form is used, either a DECB addrx or (1) must be specified; if (1) is written, the address of a DECB must be loaded into register 1 before execution of this macro instruction. Any E-form parameter overrides the corresponding parameter in the L-form. If a parameter is not specified in the L-form, it must be specified in the E-form. Required parameters for the L- and E-forms are:

Operand	L-Form	E-Form
decb	decb-symbol	decb-{addrx / (1)}
MF	MF=L	MF=E

CHECK -- Wait for and Test Completion of an I/O Request (R)

The CHECK macro instruction (for IOREQ facility) waits, if necessary, for the completion of an I/O request and detects errors and exceptional conditions. If the I/O operation is successful, the program resumes execution at the instruction after the CHECK macro instruction.

The CHECK macro instruction must be used to test for the completion of every IOREQ executed. A DECB furnished in an IOREQ must not be altered by the user until a CHECK has been issued for this DECB.

Name	Operation	Operand
[symbol]	CHECK	decb-{addrx / (1)}

decb
specifies the address of the DECB furnished in the IOREQ macro instruction that is being checked.

If (1) is furnished as the operand, the address of the DECB must have been loaded into general register 1 before the CHECK macro instruction is used.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: The CHECK macro instructions must be issued in the same order in which the associated IOREQ macro instructions were issued.

If an IOREQ results in a unit check or unit exception, the CHECK of the DECB associated with this IOREQ causes control to be given to the user's SYNAD routine specified in his data control block. If a linkage to SYNAD is executed by CHECK, all outstanding IOREQs are purged from the system. In the user-provided SYNAD routine, the user may reference the DEC field of the data control block to facilitate reissuing any of the purged IOREQs. A RETURN may be issued in a SYNAD routine that causes control to be returned to the next sequential instruction following the CHECK macro instruction that invoked the SYNAD routine.

Upon entry to the SYNAD routine, general register 1 contains the address of the DECB associated with the IOREQ involved.

When a subsequent IOREQ is executed after the SYNAD routine is invoked, the contents of the area pointed to by DCBDEC in the data control block may be changed.

If the DCBDEVD field is zero or defaulted, any unit check or unit exception causes the CHECK of the appropriate DECB to invoke SYNAD.

VCCW -- Define a Virtual Channel Command Word (O)

The VCCW macro instruction (for IOREQ facility) generates a doubleword, the virtual channel command word, that contains the proper information to inform the IOREQ macro instruction of the I/O activity requested.

Name	Operation	Operand
[symbol]	VCCW	command-{code | absexp}, data-relexp,count-absexp [,flag-({CD|CC|NCC|SCC|IOC},[SIL],[SKP])]

command
an absolute expression that specifies the hexadecimal command code. This expression's value is right justified in byte 1 of the VCCW doubleword.

The command codes, shown below, may also be supplied as a code operand. The apostrophes are part of the code and must be written if the code form of the operand is supplied.

Code Furnished in Macro	Instruction	Hexadecimal Command Code Provided
'WRITE'		01
'READ'		02
'NOP'		03
'SENSE'		04
'TIC'		08
'READBK'		0C

data
specifies the data address of the VCCW to be generated (one word).

count
specifies the count of the VCCW to be generated (two bytes).

flag
specifies which flags are to be set in the VCCW to be generated

CD - Chain Data flag
CC - Chain Command flag
SCC - Software Command Chaining flag
IOC - IORCB Chaining flag
NCC - Indicates No Command Chaining (Command chaining is default condition)
SIL - Suppress Length Indicator flag
SKP - Skip flag

PROGRAMMING NOTES: A virtual channel command word (VCCW) is a doubleword located on a doubleword boundary with this format:

Byte 0 - channel command
Byte 1 - flag byte

Bit 0 CD Chain Data flag
1 CC Chain Command flag
2 SIL Suppress Length Indicator flag
3 SKP Skip flag
4 SCC Software Command Chaining flag[1]
5 IOC IORCB Chaining flag[2]
6 Reserved
7 Reserved

Bytes 2-3 binary count field of instruction
Bytes 4-7 address in virtual storage

[1]Software command chaining causes channel end and device end associated with a command to invoke the execution of the next sequential command.
[2]See "Programming Notes," under "IOREQ."

A list of VCCWs generated by use of the VCCW macro instruction may be used to inform the IOREQ macro instruction what I/O activity is requested.

Restrictions: The list of VCCWs must conform to the following rules:

1. If any VCCW in the VCCW list has the SCC flag set,

   a. The last instruction to be executed must be the last instruction in the VCCW list. This is accomplished by having this instruction the only instruction in the list other than a TIC which does not have a CD, CC, or SCC flag set.

   b. The last instruction in the list must not be a TIC.

   c. Only the last instruction may have the IOC flag set.

2. If no VCCW in the VCCW list has the SCC flag set,

   a. An instruction executed in the VCCW list, other than a TIC, that does not have the CD or CC flag set is the last instruction executed.

   b. The last instruction in the list may have the IOC flag set only if it is the last instruction in the list to be executed.

3. The last instruction in the VCCW list must not have the CD, CC, or SCC flag set.

4. If a VCCW has the CD flag set, the following VCCW must have the same command code or be a TIC.

5. If a VCCW has the CD flag set and the following VCCW is a TIC, the TIC address must point to a VCCW with the same command code as the VCCW preceding the TIC.

6. No VCCW may have a count field of 0 unless it is a TIC.

7. The address of a VCCW incremented by the VCCW count field must not cross a page boundary.

8. The entire VCCW list must not refer to more than eight different pages of storage.

9. The VCCW list requests the supervisor to allocate space for executing a particular VCCW when an IOREQ macro instruction is issued.

   a. In the buffered IOREQ, all commands and data must be contained in one IORCB.

   b. In the nonbuffered IOREQ, all commands and page lists must be contained in the IORCB.

10. When IORCB chaining is requested, the IOC flag must be set on the last VCCW of the list (generally a NOP). This command must be the last command in the list to be executed.

If there is a question as to whether a VCCW list requires too large an area, an IOREQ macro instruction may be executed and the return code tested.

MANIPULATING ENTIRE DATA SETS

Entire data sets (rather than individual records within a data set) can be manipulated and transferred from one storage device to another. A data set can be moved from one direct access device to another, or simply to a different virtual storage area on the same direct access device. They can also be transferred from virtual storage to punched cards, printer listings, or magnetic tape devices. Several macro instructions are provided with TSS/360 data set management facilities for performing these operations. These macro instructions fall into two groups; Copying Data Sets, and Bulk O/P facilities; these groups and their related macro instructions are briefly summarized below.

COPYING DATA SETS

A user might decide to include an existing data set in a partitioned data set, to renumber the lines of an existing line data set, or to merely store an existing data set on a different device type, thereby, freeing or releasing the device on which the existing data set is stored. The functions of the CDS macro instruction, that has been provided with the TSS/360 data set management facilities to aid a user in accomplishing this type of operation, are summarized briefly below.

CDS creates copies of existing data sets or members of partitioned data sets that have been previously defined to the system and reside on direct access or magnetic tape volumes. It also creates copies of line data sets with renumbered lines. The copy is placed into a new data set. Both the new data set and the existing old data set must be previously defined to the system via issuance of the DCB macro instruction and the DDEF macro instruction (or command). The old data set does not, however, have to be opened by the user. It is opened automatically by the CDS routine.

A detailed explanation of the above macro instruction and the format in which it may be specified is shown below. Further information pertaining to the manipulation of an existing data set and the CDS macro instruction can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C-28-2032.

CDS -- Copy Existing Data Set (S)

The CDS macro instruction copies a data set or a member of a partitioned data set. In addition, it may renumber the lines of a line data set. The resulting new data set is assigned the data set name furnished (as an operand) by the user. A copy of a member may be specified either as a new member of a partitioned data set, or as a new data set by itself. A virtual storage data set may be copied as a member of a partitioned data set.

Name	Operation	Operand
[symbol]	CDS	oplist-{text / addr}

oplist
specifies the list of operands. They are:

Oplist
dsname$_1$-symbol,dsname$_2$-symbol[,E] [,[line-integer] [,increment-integer]]

dsname$_1$
specifies the data set name of the data set being copied. It must be cataloged or have been defined in a DDEF macro instruction or command.

This operand can be specified as the fully qualified name of: a nonpartitioned data set, a member of a partitioned data set, or a nonpartitioned generation of a generation data group (identified by absolute generation name or relative generation number).

dsname$_2$
specifies the data set name assigned to the copy of the data set. It must have been defined in a DDEF macro instruction or command unless a member of a cataloged partitioned data set is specified.

This operand can be specified as the fully qualified name of: a nonpartitioned data set, a member of a partitioned data set, or a nonpartitioned generation of a generation data group (identified by absolute generation name or relative generation number).

E
specifies that the original data set or data set member is to be erased after being copied. E applies only to data sets on direct-access devices. If a shared data set is to be copied and then erased, unlimited access to the data set must have been permitted.

line
specifies the starting line number of the data set copy if it is a line data set and renumbering is desired. The number consists of three to seven digits, the last two of which should be zero. An all-zero starting line number is invalid.

Default: If increment is also defaulted, line numbering is not performed. If increment is not defaulted, the starting line number of the copy data set will be 100.

increment
specifies the value by which line numbers in the data set copy (if it is a line data set) are to be incremented when renumbering is desired. It consists of three to seven digits, the last two of which should be 0. An all-zero increment is invalid.

Default: If the starting line number is also defaulted, line numbering is not performed. If the starting line number is not defaulted, an increment of 100 is assumed.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: The CDS macro instruction is restricted to data sets on direct-access or magnetic-tape volumes. Data set organization is not altered by the use of a CDS macro instruction. The dsname$_1$ and dsname$_2$ data sets must be defined with the same data set organization and record format. For example, the copy of a physical sequential data set has

physical sequential organization, even though the device type may be changed. A VISAM data set can be copied as VSAM and vice versa.

The user may specify a VISAM organization in the CDS macro instruction for a data set copy even though the original data set organization is VSAM. In this case, each record of the original data set must contain a key. In addition, the user should define -- in the DDEF macro instruction or command for the data set copy -- the key length (KEYLEN), padding (PAD), and record key displacement (RKP) values. If he does not provide these values, no copy is made.

An entire partitioned data set cannot be copied with one CDS macro instruction. Each member must be copied individually. A separate macro instruction, specifying the data set name and member name, must be issued for each member.

This macro instruction cannot be used to copy program modules because program modules have format-U (undefined) records.

The user can copy only those data sets that belong to him or those to which he has been given access.

At completion of execution of the CDS macro instruction, the low-order byte of register 15 contains one of the following codes:

Code (Hexadecimal)	Significance
00	Normal
04	Invalid input parameters
08	Name of original data set not in catalog or task definition table (TDT)
0C	Data set not in catalog and no DDEF macro instruction or command has been executed for it
10	JFCB for original data set not consistent with JFCB for new data set
14	Member name not given for partitioned data set
18	User does not have write access for new data set
1C	Original data set not VAM or SAM
20	Data set not on direct-access; command ignored
24	New data set member name already exists in POD
28	Data set copied. Old data set not erased; user does not have proper access
2C	Data set copied. New data set not renumbered; not a line data set
30	Data set copied and renumbered. Old data set not erased; user does not have proper access
34	Data set copied and original erased. New data set not renumbered; not a line data set
38	Data set copied; new data set not renumbered, and old data set not erased

L- AND E-FORM USE: The oplist operand is required in the L-form of this macro instruction and is not permitted in the E-form. Only the text form of the operand can be used in the L-form of the macro instruction.

EXAMPLES: IN EX1, THE OPLIST IS PRESENTED AS A CHARACTER STRING. In EX2, an address designates the oplist.

```
EX1     CDS     'DATASET,U'
EX2     CDS     OPLISTC
```

BULK OUTPUT FACILITIES

The bulk output facilities allow a user to transfer entire data sets from virtual storage to punched cards, printer listings, or magnetic tape devices (for off-line printing). These facilities provide a user with three macro instructions, the print (PR), punch (PU), and write tape (WT) macro instructions, which enable him to accomplish these transfers. These macro instructions are to be issued in a user program on closed SAM, VSAM, and VISAM data sets only. Although VPAM data sets or members cannot employ these macro instructions, the members of the VPAM data set could first be copied with a CDS macro instruction (or command) into new VSAM of VISAM data sets and then be operated on by these macro instructions. Execution of these macro instructions cause requests for particular output operations to be set up as independent nonconversational tasks, places the requested task on a bulk output queue, and returns to the user's problem program. The user can then continue processing other data sets (and could even terminate) while the output task waits to be, or is being, executed. The bulk output macro instructions are briefly described below.

PR causes a specified data set to be listed on a high speed on-line printer and optionally erases it from the user's catalog when the printing has been finished. Line spacing on the printed output can also be indicated by the user. The print operation takes place as an independent nonconversatioal task.

PU causes a specified data set to be listed on a high speed on-line punch and optionally erases it from the user's catalog when the punching is finished. Stacker selection can also be indicated by the user. The punch operation takes place as an independent nonconversational task.

WT writes a specified data set on magnetic tape in proper format for subsequent off-line printing and optionally erases it from the user's catalog when the writing is finished. The write tape operation takes place as an independent nonconversational task.

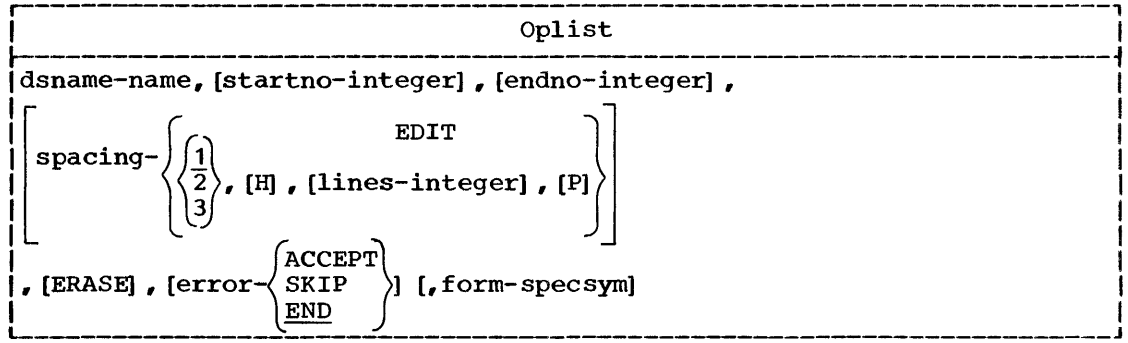
Detailed explanations of each of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to bulk output facilities and the related macro instructions can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

PR -- Print a Data Set (S)

The PR macro instruction causes the specified data set to be listed in nonconversational mode on a high-speed line printer and, optionally, erases it from the catalog when printing is finished. The specified data set is printed as it stands, with no code conversions; i.e., the data set must be in EBCDIC character codes so proper printer graphics will be used in printing. If the data set resides on a seven-track tape, any character adjustments required to ensure data validity are made by the system.

Name	Operation	Operand
[symbol]	PR	oplist-{text / addr}

oplist
specifies the list of operands supplied for the PR macro instruction:



dsname
specifies the name of the data set to be printed. The data set name either must have been previously defined by a DDEF command or macro instruction, or the dsname must be in the catalog.

This operand can be specified as the fully qualified name of a nonpartitioned data set or of a nonpartitioned generation of a generation data group (identified by absolute generation name or relative generation number).

startno
specifies the byte number at which printing is to start for each data set record. The number consists of one to six decimal digits. If this operand is not specified, printing starts with the first byte of each record.

endno
specifies the byte number at which printing is to stop for each data set record. This end byte is printed. If this operand is not specified, printing continues to the last byte of each logical record or until the printer line length is reached, whichever occurs first.

spacing
specifies the number of lines to be skipped.

EDIT
indicates that the line spacing is controlled by a control character in the first byte position of each data set logical record. This control character is programmer-supplied and may be in USASI (USASCII) or machine code, but must be in the same code throughout the data set. (Refer to Appendix D.)

Note: If EDIT is selected, the line skipping, header, lines per page, and page numbering options cannot be specified in this macro instruction.

1 - indicates skip 1 line
2 - indicates skip 2 lines
3 - indicates skip 3 lines

H
specifies that the first logical record of the data set is to be repeated on each print page as a header line. The first 132 bytes or the first record, whichever is smaller, will be used as the header.

lines
specifies the number of lines to be printed on a page. The number of lines is specified as a one-to-four digit decimal number. The maximum number of lines per page is determined by the printer form being used. If not specified, 54 lines are printed on each page.

P
specifies that page numbering is to be performed. If P is not specified, page numbering is not performed.

ERASE
specifies that the cataloged data set is to be erased from the catalog after the print operation is finished. If the data set is shared and is currently being used by another user a diagnostic will be issued and the erase will be ignored.

error
specifies the action to be taken if an uncorrectable error is encountered while reading a data set record. This option applies only if the data set to be printed is on tape. The options are:

ACCEPT - the error record is accepted.
SKIP - the error record is skipped.
END - the print operation is terminated.

form
specifies the form number of the printer paper to be used. It consists of one to six characters. The installation's standard printer form, defined at system generation time, is used if this operand is not specified.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: The PR macro instruction processes data sets that were created by using the basic sequential, queued sequential, virtual sequential, or virtual index sequential access method. The data set name may or may not be in the catalog. If not, it is placed in the catalog until printing is completed and is then erased. If the data set name is in the catalog, the ERASE option can be used to erase the data set after printing is completed. A basic sequential or queued sequential data set must reside on magnetic tape. A virtual sequential or virtual index sequential data set must not contain format-U records.

If the data set to be printed was created via the DATA command, the first byte of each record contains an indicator for the origin of the record. Unless the startno operand is specified, this byte is printed as part of the record upon issuance of the PR macro instruction. In such a case, if the record was originally entered through a card reader, the indicator byte will be printed as a C. If it was entered through a terminal, the byte will be printed as a blank character. When the startno operand is specified as 2 or greater, the indicator byte is bypassed and is not included as part of the printed record.

Invalid print characters appear as blanks in the output. If a read error occurs, the data set record causing the read error is output in hexadecimal on the SYSOUT data set regardless of the error option, if any, selected by the user.

At completion of execution of the PR macro instruction, register 1 contains the address of the batch sequence number assigned to the non-conversational task established by this macro instruction; the low-order byte of register 15 contains one of the following codes:

Code	Significance
0	PR request was accepted.
All other codes	Register 15 contains a two-byte message number. These messages are described under "Prompting Messages" and "Diagnostic Messages" in Command Language User's Guide.

L- AND E-FORM USE: The oplist operand is required in the L-form of this macro instruction and is not permitted in the E-form. Only the text form of the operand can be used in the L-form of the macro instruction.

EXAMPLES: In EX1, the oplist is presented as a character string. In EX2, a symbolic address designates the oplist.

```
EX1     PR      'DSNAME1,02,120,1'
EX2     PR      LSTTAG
```

Since EX2 specifies an address, the user has provided the oplist string at location LSTTAG. When the macro instruction is executed, the necessary alphameric characters must be available in the string.

PU -- Punch a Data Set (S)

The PU macro instruction causes a specified data set to be punched onto cards in nonconversational mode on a high-speed punch and, optionally, to be erased from the catalog when punching is finished. Any contiguous field of up to 80 bytes can be punched from each input record of an EBCDIC data set. The specified data set is punched as it stands, with no code conversions.

Note: Up to 160 bytes per card can be punched in a special column binary format, where bits 0 and 1 of each byte are ignored.

Name	Operation	Operand
[symbol]	PU	oplist-{text / addr}

oplist
specifies the list of operands supplied for the PU macro instruction:

Oplist
dsname-name,[BINARY] [startno-integer],[endno-integer],[select-{1 / 2 / 3 / EDIT}], [ERASE] [,form-specsym]

dsname
specifies the name of the data set to be punched. The data set name either must be previously defined by a DDEF macro instruction or command, or must be in the catalog.

This operand can be specified as the fully qualified name of a nonpartitioned data set or of a nonpartitioned generation of a generation data group (identified by absolute generation name or relative generation number).

BINARY
specifies punching in column binary format. If not specified, punching is in EBCDIC format.

startno
specifies the byte number at which punching is to start for each data set record.

If this operand is not specified, punching starts with the first byte of each record.

endno
specifies the byte number at which punching is to stop for each data set record. This end byte is punched. If this operand is not specified, punching continues to byte 80 (or, in binary, to byte 160) or to the end of the record, whichever occurs first.

select
specifies the stacker select or edit option:

1 - indicates pocket number P1
2 - indicates pocket number P2
3 - indicates pocket number P3

EDIT - indicates that the first byte of each data set logical record contains a control character for stacker selection. This control character is user-supplied and may be in USASI (USASCII) or machine code, but must be in the same code throughout the data set. (See Appendix D.)

ERASE
specifies that the cataloged data set is to be erased from the catalog after the punch operation is finished. If the data set is shared and is currently being used by another user, when this option is processed, a diagnostic will be issued and the erase will be ignored. If ERASE is not specified, no erasure is made.

form
specifies the card form number of the cards to be used for punching. The form number may have one to eight characters. If this operand is not specified, the installation's standard card, as established at system generation, is used.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: The PU macro instruction processes data sets that were created by using either the virtual sequential or virtual index sequential access method. The data set name may or may not be in the catalog. If not, it is placed in the catalog until punching is completed and is then erased. If the data set name is in the catalog, the ERASE option can be used to erase the data set after punching is completed.

If a data set to be punched was created via the DATA command, the first byte of each record contains an indicator for the origin of the record. Unless the startno operand is specified, this byte is punched as part of the record upon issuance of the PU macro instruction. In

such a case, if the record was originally entered through a card reader, the indicator byte will be punched as a C. If it was entered through a terminal, the byte will be punched as a blank character. When the startno operand is specified as 2 or greater, the indicator byte is bypassed and is' not included as part of the punched record.

Since the READ CARDS command prefixes a line number automatically to each record of a VISAM data set read from cards, any VISAM data set that is to be read from cards should not contain line numbers. Therefore, if an existing VISAM line data set is to be punched on cards and later recreated by reading those cards with a READ CARDS command, the user should be careful to punch out the stored VISAM data set without including line numbers.

Invalid characters appear as blanks when EBCDIC records are punched. If a read error occurs, the record in question is not punched, but is written in hexadecimal on SYSOUT.

At completion of execution of the PU macro instruction, register 1 contains the address of the batch sequence number assigned to the nonconversational task established by this macro instruction; the low-order byte of register 15 contains one of the following codes:

Code	Significance
0	PU request was accepted.
All other codes	Register 15 contains a two-byte message number. These message numbers are described under "Prompting Messages" and "Diagnostic Messages" in Command System User's Guide.

L- AND E-FORM USE: The oplist operand is required in the L-form of this macro instruction and is not permitted in the E-form. Only the text form of the operand can be used in the L-form of the macro instruction.

EXAMPLES: In EX1, the oplist is presented as a character string. In EX2, a symbolic address designates the oplist.

```
EX1     PU      'DSNAM2,,020, 99,,ERASE'
EX2     PU      CDTAG
```

Since EX2 specifies an address, the user has provided the oplist string at location CDTAG. When the macro instruction is executed, the necessay alphameric characters must be available in the string.

WT -- Write a Data Set on Tape for Off-Line Printing (S)

The WT macro instruction edits and writes the specified data set on magnetic tape in nonconversational mode for subsequent off-line printing and, optionally, erases it from the catalog when writing is finished. The output is written on 9-track tape in odd parity with standard System/360 labels. Each input data set record is written as a logical record, or print line, on tape in proper format for off-line printing; records are blocked, if requested. The maximum blocked record length is 32,767 bytes.

Name	Operation	Operand
[symbol]	WT	oplist-{text / addr}

oplist
specifies the list of operands supplied for the WT command. They are:

Oplist
dsname$_1$-name, dsname$_2$-name , [volume-alphnum] , [factor-integer] , [startno-integer] , [endno-integer] [, spacing-{EDIT / (1 / 2 / 3)} , [H] , [lines-integer] , [P]] [,ERASE]

dsname$_1$
specifies the name of the data set to be written on tape in print format. The data set name either must be previously defined by a DDEF macro instruction or command, or must be in the catalog.

This operand can be specified as the fully qualified name of a nonpartitioned data set or a nonpartitioned generation of a generation data group (identified by absolute generation name or relative generation number).

dsname$_2$
specifies the data set name under which the data set is to be cataloged as it resides on the output tape. The user must specify dsname$_2$ or the task may be abended.

This operand can be specified as the fully qualified name of a nonpartitioned data set or a nonpartitioned generation of a generation data group (identified by absolute generation name or relative generation number).

volume
specifies the volume ID number of the output tape. The ID number consists of one to six alphameric characters. If volume is not specified, a scratch tape is used.

factor
specifies the blocking factor of the output tape. The factor consists of one to three decimal digits; the maximum blocking factor permitted is 246. If the blocking factor is not specified, a blocking factor of 30 is assumed.

startno
specifies the byte number at which tape writing is to start for each data set logical record. The number consists of one to six decimal digits. If the operand is not specified, writing starts with the first byte of each record.

endno
specifies the byte number at which printing is to stop for each data set record. This end byte is written. If this operand is not

specified, writing continues to the last byte of each logical record or until the printer line length is reached, whichever occurs first.

spacing
specifies the number of lines to be skipped.

EDIT - indicates that the line spacing is controlled by a control character in the first byte position of each data set logical record. This control character is user-supplied and may be in ASA (USASCII-II) or machine code, but must be in the same code throughout the data set (Refer to Appendix D.)

1 - indicates skip 1 line
2 - indicates skip 2 lines
3 - indicates skip 3 lines

H
specifies that the first logical record of the data set is to be repeated on each print page as a header line. The first 132 bytes or the first record, whichever is smaller, will be used as the header.

lines
specifies the number of lines to be printed on a page. The number of lines is specified as a one-to-four digit decimal number. The maximum number of lines per page is determined by the printer form used for the off-line printing of the data set. If not specified, 54 lines are printed on each page.

P
specifies that page numbering is to be performed. If P is not specified, no page numbering is performed.

ERASE
specifies that the cataloged data set is to be erased from the catalog after the tape-writing operation is finished. If the data set is shared and is currently being used by another user, a diagnostic will be issued and the erase will be ignored. If ERASE is not specified, no erasure is made.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: The WT macro instruction processes input data sets that were created by using either the virtual sequential or virtual index sequential access method. The tape data set is created by using the basic sequential access method. This output tape is written in odd parity with standard Time Sharing System/360 labels. If the input data set is not in the catalog, it is placed in the catalog until writing is completed and is then erased. If the data set name is in the catalog, the ERASE option can be used to erase the data set after writing is completed.

If a data set to be written on tape was created via the DATA command, the first byte of each record contains an indicator for the origin of the record. Unless the startno operand is specified, this byte is written as part of the record upon issuance of the WT macro instruction. In such a case, if the record was originally entered through a card reader, the indicator byte will be written as a C. If it was entered through a terminal, the byte will be written as a blank character. When the startno operand is specified as 2 or greater, the indicator byte is bypassed and is not included as part of the written record.

No more than one print line can be written from a single data set record. If a read error occurs, the record in question is written in hexadecimal form on SYSOUT.

At completion of execution of the WT macro instruction, register 1 contains the address of the batch sequence number assigned to the non-conversational task established by this macro instruction; the low-order byte of register 15 contains one of the codes given below.

Code	Significance
0	WT request was accepted.
All other codes	Register 15 contains a two byte message number. These message numbers are described under "Prompting Messages" and "Diagnostic Messages" in Command Language User's Guide.

L- AND E-FORM USE: The oplist operand is required in the L-form of this macro instruction and is not permitted in the E-form. Only the text form of the operand can be used in the L-form of the macro instruction.

EXAMPLES: In EX1, the oplist is presented as a character string. In EX2, a symbolic address designates the oplist.

```
EX1     WT      'OLDNAME,NEWNAME'
EX2     WT      TAPTAG
```

Since EX2 is given an address, the user has provided the oplist string at location TAPTAG. When the macro instruction is executed, the necessary alphameric characters must be available in the string.

CATALOG DATA SET ATTRIBUTES

Once the attributes of a data set have been described to the system via user (or system) issuance of the DCB macro instruction, the DDEF macro instruction (or command), or any of the other available sources for attributes, certain attributes in the data set description should be stored within the system so that it can subsequently be located by using only its name. The CAT macro instruction has been provided with the TSS/360 data management facilities for recording such attributes in a user's catalog.

Such attributes are automatically cataloged at the initial DDEF time by the system for all public VAM data sets; however, for private data sets, the user must request that such attributes be recorded in the catalog by issuing a CAT macro instruction. Catalog entries for both public and private data sets can also be altered by issuing CAT. These entries can be subsequently deleted from a user catalog by issuing a delete (DEL) macro instruction. The CAT and DEL macro instructions are briefly described below.

CAT — record specific data set attributes as catalog entries for all private data sets, groups of data sets, all generations of a generation data group, and for partially qualified names (all data sets with the same partial name). It also alters calalog entries of both public and private data sets and recatalogs, expanding and contracting multivolume SAM data sets.

DEL — deletes one or more catalog entries for a data set or group of data sets from the user catalog. For generation data groups, entries for all generations are deleted; for partially qualified data sets names, all catalog entries with the same initial name component are deleted.

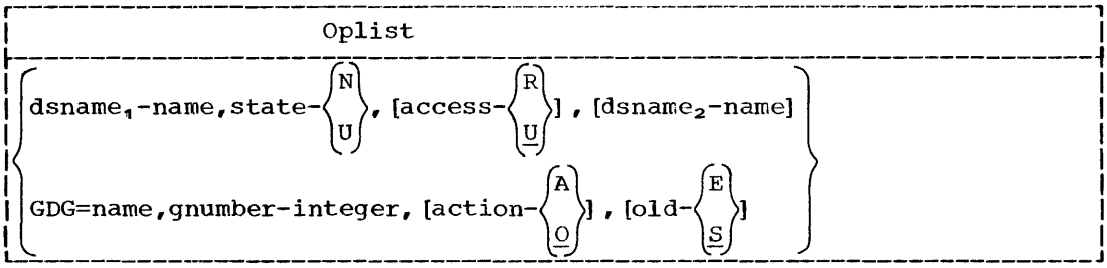
Detailed explanations of each of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to cataloging data set attributes and its related macro instructions can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

CAT -- Create or Change Catalog Entry (S)

The CAT macro instruction can create catalog entries for all private data sets (i.e., all public VAM data sets are immediately cataloged at initial DDEF time by the system) or it can be used to create a catalog index for a generation data group or to catalog a data set as a new generation of an existing generation data group. The CAT macro instruction can also be employed to alter the catalog entry for both private and public data sets (i.e., to rename a data set, or to change the access qualification, etc.) or to recatalog expanding and contracting multivolume SAM data sets.

Name	Operation	Operand
[symbol]	CAT	oplist-{text / addr}

oplist
specifies the list of operands. They are:



dsname$_1$
specifies the name of a SAM or a private VAM data set defined in a DDEF macro instruction or command, or specifies a cataloged public or private data set name. The data set must reside on a direct-access or magnetic tape volume.

This operand can be specified as:

- The fully qualified name of a partitioned or nonpartitioned data set or a partitioned or nonpartitioned generation data group (identified by absolute generation name or relative generation number).
- The partially qualified name of any data set other than a generation data group.

state
specifies the updating of an existing catalog entry, or the creation of a new catalog entry:

N - new creation
U - update

access
specifies the owner access qualification for the data set:

R - read-only access
U - unlimited access

If this operand is not specified, unlimited access is assumed. This default is valid only if a new catalog entry is being made; otherwise, no change is made to the access qualification. If R is specified, the owner may not write into his data set but he may erase his data set.

dsname$_2$
specifies the new name for the data set. This parameter is necessary only if the currently defined name of the data set is to be changed. The dsname may have a relative generation number appended.

This operand can be specified as:

- The fully qualified name of a partitioned or nonpartitioned data set or a partitioned or nonpartitioned generation data group (identified by absolute generation name or relative generation number).
- The partially qualified name of any data set other than a generation data group.

GDG
specifies the name of a new generation data group.

gnumber
specifies the number of generations to be maintained in the generation data group.

action
specifies the action to be taken when the n+1 generation is being cataloged in the generation data group:

A - all previous generations are to be removed from the catalog.
O - only the oldest generation is to be removed.

If this operand is omitted, O is assumed.

old
specifies the disposition of old generations deleted from the catalog. This applies to private volumes only; data sets on public volumes are always erased when uncataloged.

E - erase external storage belonging to old generation data group members.
S - save old generation data group members.

If this operand is omitted, S is assumed.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

The expansion and contraction of SAM or VAM multivolume data sets are handled differently. Whenever an existing SAM multivolume data set has been reduced or expanded in size, thereby residing on fewer or more volumes, it must be recataloged by the user with the CAT macro instruction. For VAM, the system automatically recatalogs multivolume data sets which expand or contract.

PROGRAMMING NOTES: When $dsname_1$ is given, a new entry is made in the catalog if the N option was specified. When the U option is given, the catalog entry is updated with the requested changes to the data set name and/or access qualifier. In addition, when $dsname_2$ is supplied, a change is made to the name in the data set labels (DSCBs) on the volumes containing the data set. This step is omitted if the volumes are on tape.

If the GDG keyword is specified, an index is created for a new generation data group using the parameters supplied. If the generation data group is already cataloged, no updating is possible.

If the dsname is specified with a member name, only the dsname itself is used; the member name is removed.

If the user wants to change the definition information for a cataloged data set, he must issue a new DDEF macro instruction, containing the new information, and for SAM and private VAM data sets, a CAT macro instruction. For public data sets issuance of the DDEF will cause the new information to be automatically cataloged by the system. This effectively updates the entry for that data set. The information in the CAT macro instruction may, of course, be changed merely by issuing a CAT macro instruction with "update" indicated.

For private data sets only, the owner of a generation data group is allowed to catalog generations of that group. Sharers, regardless of their level of access, are not permitted to do this.

Generations of a generation data group that reside on private storage can be saved by the user even after they are uncataloged.

At completion of execution of the CAT macro instruction, the low-order byte in register 15 contains one of the following codes:

Code (Hexadecimal)	Significance
00	Cataloging accomplished as requested
04	Name changed in catalog but not on one or more volumes
08	Invalid element in input string
0C	Volume not available, or wrong kind of storage
10	Data set name not unique, already in catalog
14	No volume of data set mounted; cannot catalog
18	ABEND request

L- AND E-FORM USE: The oplist operand is required in the L-form of this macro instruction and is not permitted in the E-form. Only the text form of the operand can be used in the L-form of the macro instruction.

EXAMPLES: In EX1, the oplist is presented as a character string. In EX2, an address designates the oplist.

```
EX1     CAT       'DATASET,U,U'
EX2     CAT       OPLISTC
```

DEL -- Delete Catalog Entry (S)

The DEL macro instruction deletes one or more catalog entries for a data set or group of data sets. When a generation data group name is supplied, the macro instruction deletes the catalog entries for all generations in that group. Similarly, a partially qualified data set name results in catalog entries being deleted for all data sets with the same initial name component.

Name	Operation	Operand
[symbol]	DEL	dsname-{name / addr}

dsname
specifies the name of the data set whose catalog entry is to be deleted.

This operand can be specified as:

- The fully qualified name of: a partitioned or nonpartitioned data set, or a partitioned or nonpartitioned generation of a generation data group (identified by absolute generation name or relative generation number).
- The partially qualified name of any type of data set, including a generation data group.

See "Oplist Operands" in Section I for using the addr form of this operand.

If the data set is not shared, it must reside on a private volume; the dsname may be the sharer's name for a data set owned by another user.

CAUTION: This macro instruction deletes the catalog entries for data sets on private volumes only. A macro instruction that attempts to uncatalog data sets residing in public storage is ignored and a diagnostic message is produced if in conversational mode. Only the ERASE command can be used to remove such data sets from the system. However, the DEL macro instruction can be used to delete a sharing descriptor from the sharer's catalog.

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: Data sets on public volumes must be erased if they are to be uncataloged. The user must, therefore, use the ERASE command to remove those data sets from the system except when he is a sharer.

At completion of execution of the DEL macro instruction, the low-order byte of register 15 contains one of these codes:

Code (Hexadecimal)	Significance
00	Valid return code
04	Not class D nor batch monitor
08	Invalid return from NEXTPAR
0C	Invalid dsname (input preceded by left parentheses) - NEXTPAR
10	No dsname supplied after verb
14	Return code from CHECKDS was not divisible by four
18	Data set not cataloged nor in task definition table (TDT)
20	Partitioned data set not in POD
24	Data set not cataloged
28	Data set on a public volume
2C	Data set name is a member of a partitioned data set
30	User does not own nor share data set
34	Sharer does not have unlimited access to data set
38	Data set not cataloged
3C	DDEF return code indicating dsname is unknown to the catalog; hence, no JFCB created
40	Attention interruption

L- AND E-FORM USE: The oplist operand is required in the L-form of this macro instruction and is not permitted in E-form. Only the text form of the operand can be used in the L-form of the macro instruction.

DISCONNECTING A DATA SET FROM THE SYSTEM

Just as connecting a data set to the system tells the system a user is ready to process that data set, disconnecting a data set from the system tells the system a user has finished processing a data set. Thus, when a user has finished his processing of a data set he must inform the system by disconnecting the data set from the system. A data set may be permanently or temporarily (for BSAM only) disconnected from the system. The CLOSE macro has been provided with TSS/360 data set management facilities to allow the user to disconnect a data set from the system in this manner. Descriptions of the permanent and temporary close are briefly described below.

CLOSE (permanent close for all access methods) locates the description of data set attributes currently recorded in the data control block defined by a DCB macro instruction issued for that data set and (for BSAM only) determines if all I/O requests have been satisfied. If they have not been satisfied, the CLOSE routine waits until they are satisfied before proceeding. Output data set labels are then created, volumes are positioned as specified by the user and the control blocks (DCB and JFCB) containing descriptions of the data set's attributes are restored to their pre-open status thereby logically disconnecting that data set from further system processing and preventing further user access to the data set. A subsequent OPEN macro instruction must be issued for this data set if additional processing is required.

CLOSE (T) (temporary close for BSAM data sets only) same as the standard CLOSE macro instruction except that the fields of the control blocks (DCB and JFCB) are not restored to their pre-open status; the data control blocks remain in an open status and additional processing may be performed on that data set without issuing another OPEN macro instruction. The temporary close is useful as a simple way of repositioning a volume for subsequent processing.

Detailed explanations of each of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to disconnecting a data set from the system and the related CLOSE macro instruction can be found in IBM/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

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.

During the execution of CLOSE, user's trailer label routine, if supplied, will be given control (BSAM and QSAM only). (Refer to Appendix A.)

Name	Operation	Operand
[symbol]	CLOSE	({dcb-addr, [opt-{REREAD / LEAVE}]} ,...) [,TYPE=T]

dcb (all access methods)
specifies address of data control block opened for data set that

is to be permanently or temporarily disconnected (closed) from the system. If more than one data control block is specified, two commas must be placed between each to indicate the omission of the positioning option, even though it is applicable to BSAM and QSAM only.

opt (BSAM and QSAM)
specifies volume repositioning that is to be performed as a result of closing. Its values and meanings are:

REREAD - positions current volume to process data set again.

LEAVE - positions current volume to logical end of data on the volume. This value is assumed if the opt operand is omitted.

The opt operand is applicable to volume disposition of magnetic tape devices only; it is ignored for other devices.

TYPE=T (BSAM ONLY)
is written as shown. It indicates that labels are created and volumes are positioned, but the fields of the data control block are not altered. The data set can be processed without issuing another OPEN macro instruction. If TYPE=T is designated, it applies to all of the associated data control blocks.

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 OPEN status. If CLOSE (TYPE=T) is issued for a direct-access data set volume, a regular CLOSE is executed.

CAUTION: The following errors cause the results indicated.

Errors	Result
Permanently or temporarily closing a data control block that is not open	No Action
Temporarily closing (TYPE=T) a data control block that has not been opened for BSAM	No Action
Permanently closing when the dcb operand does not specify the address of a data control block	Task terminated
Temporarily closing (TYPE=T) when the dcb operand does not specify the address of a data control block	Unpredictable
Permanently closing a dcb containing an invalid DSORG specification	Task terminated

If this macro instruction is included in a module that is declared privileged through use of the DCLASS macro instruction, the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: Any number of data control block addresses and associated options (BSAM and QSAM) 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, which is more efficient than to close them individually.

VAM only

The CLOSE macro instruction releases any sharing interlocks set for the data set. Rules for sharing VAM data sets are given in APPENDIX K.

If more than one data control block is specified in a CLOSE macro instruction for VAM data sets, two commas must be placed between each to indicate the omission of the repositioning operand, which is applicable to BSAM only.

BSAM and QSAM only

The CLOSE (TYPE=T) macro instruction may be used to disconnect temporarily, from the problem program, one or more data sets if they reside on magnetic tape. 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 volumes are repositioned as specified in this macro instruction.

Magnetic tape positioning varies depending on the options chosen in OPEN and CLOSE (TYPE=T) macro instructions. Table 4 relates the options in macro instructions to the repositioning of tape volumes.

For magnetic tape, positioning varies depending on whether the data set uses labels. Table 3 defines a final position number for labeled and unlabeled tapes and Table 4 relates the options chosen in OPEN and CLOSE macro instructions to positioning of tape volumes.

User trailer-label exits are taken for a data set processed for INOUT or OUTIN if last operation was a WRITE. No user trailer label exits are taken if last operation was a READ.

Table 3. Final Magnetic-Tape Positions

Position	Labeled Tape	Unlabeled Tape
1	Preceding data set header label group on current volume	Preceding first data block of portion of data set resident on current volume
2	Following tape mark that terminates trailer label group of data set on current volume	Following tape mark that terminates last data block of portion of data set resident on current volume

Table 4. Factors Determining Magnetic-Tape Positioning For BSAM and QSAM

Opt$_1$ of OPEN Specified as	Other Factors Influencing Positioning	Direction of Last Input Operation	Positioning as Specified by Opt in CLOSE	
			LEAVE	REREAD
OUTPUT	--	Not applicable	Position 2	Position 1
OUTIN (BSAM only)	--	Not determining factor		
INOUT (BSAM only)	At least one WRITE operation in this data set	Not determining factor		
INPUT	--	Forward		
INOUT (BSAM only)	No WRITE operation executed in this data set	Forward		
RDBACK	--	Forward		
INPUT	--	Backward	Position 1	Position 2
INOUT (BSAM only)	No WRITE operation executed on this data set	Backward		
RDBACK		Backward		

If the data set resides on a magnetic tape, the following concerns the writing of trailer labels:

1. If data set was opened for OUTIN or INOUT and the last I/O operation was a WRITE, then CLOSE or CLOSE (TYPE=T) both cause trailer labels to be written. If CLOSE (TYPE=T) is issued, additional READ or WRITE macro instructions are accepted without issuing a new OPEN macro instruction.

2. If data set was opened for OUTIN or INPUT and the last I/O operation was a READ, and then CLOSE or CLOSE (TYPE=T) was issued, additional READ and WRITE macro instructions are accepted without a new OPEN macro instruction being given.

3. If data set was opened for OUTPUT, A CLOSE or CLOSE (TYPE=T) both cause trailer labels to be written. If CLOSE (TYPE=T) is issued, additional WRITE macro instructions are accepted without a new OPEN macro instruction being given.

4. If data set was opened for INPUT or RDBACK, a CLOSE or CLOSE (TYPE=T) does not cause trailer labels to be written. If CLOSE (TYPE=T) is issued, additional READ macro instructions are accepted without a new OPEN macro instruction being given.

L- AND E-FORM USE: L- and E-forms of the CLOSE macro instruction are allowed. The TYPE=T often is not permitted in the E-form. The E-form of the macro instruction may specify any parameters; however, parameters specified in the E-form will overlay those specified in the L-form. The E-form may not specify more DCB operands than are specified in the

corresponding L-form. The format of the parameter list generated by the CLOSE macro instruction is described in Appendix L.

For example:

```
JOE         CLOSE       (ADCB,,BDCB,,),MF=L
TERI        CLOSE       (,,PRODCB,,AXDCB),MF=(E,JOE)
```

When the E-form macro instruction is executed, the data control block PRODCB replaces the data control block BDCB in the parameter list, and the data control block AXDCB is added to the parameter list in the position reserved by the two commas following BDCB in the L-form. Thus, data control blocks with symbolic address ADCB, PRODCB, and AXDCB are closed.

EXAMPLES:

for BSAM or QSAM:

EX1 closes the data set associated with data control block INVEN with no repositioning. EX2 closes the two data sets associated with data control blocks INVEN and REPORT with different options. EX3 closes data sets associated with two data control blocks. Since opt is omitted in EX3, volume disposition is defaulted as LEAVE. EX4 generates a parameter list for closing INVEN, and EX5 closes INVEN.

```
EX1         CLOSE       (INVEN,LEAVE)
EX2         CLOSE       (INVEN,LEAVE,REPORT,REREAD)
EX3         CLOSE       (INVEN,,MASTER)
EX4         CLOSE       (INVEN,LEAVE)
EX5         CLOSE       MF=(E,EX4)
```

for VAM:

EX1 closes data sets associated with two data control blocks. EX2 generates a parameter list for closing INVEN, and EX3 closes INVEN.

```
EX1         CLOSE       (INVEN,,MASTER)
EX2         CLOSE       (INVEN),MF=L
EX3         CLOSE       MF=(E,EX2)
```

REMOVING A DATA SET FROM THE SYSTEM

When a user no longer has any use for a particular data set, it is wasteful to leave such a data set recorded on a device that could be used for storing new data into the system. For this reason, TSS/360 data set management facilities provide a user with two macro instructions, ERASE and REL, which can physically remove a data set from the system and release the input/output devices on which they had been recorded for future system use.

ERASE for data sets recorded on direct access devices this macro instruction erases the data set from the device by erasing the direct access storage assigned to the data set. It also removes any catalog entry for the erased data set that may have been established in a user's catalog (via previous system or user issuance of a CAT macro instruction).

REL in effect, cancels or erases a previously defined public or private data set from the system by deleting the attribute specifications previously defined to the system by issuance of a DDEF macro instruction. For private data sets, the input/output devices associated with it are released to a system resource pool. It can also be used to release all data sets of a concatenated data set or to remove a users JOBLIB.

Detailed explanations of each of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to physically removing a data set from the system and the related macro instructions can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

ERASE -- Remove a Data Set from Direct-Access Storage (S)

The ERASE macro instruction erases the direct-access storage assigned to a data set. In addition, it removes the entry for a cataloged data set from the catalog.

Name	Operation	Operand
[symbol]	ERASE	dsname-{text / addr}

dsname
specifies the name of any data set residing on direct-access storage. The data set name must be cataloged or must already be defined within the current task.

This operand can be specified as:

- The fully qualified name of: a partitioned or nonpartitioned data set, a member of a partitioned data set, or a partitioned or nonpartitioned generation of a generation data group (identified by absolute generation name or relative generation number).
- The partially qualified name of any type of data set, including a generation data group.

If the text option is selected, the data set name, enclosed in apostrophes, is written as the operand; if the addr option is selected, the operand specifies the location of the data set name.

If the data set name does not involve a member name, the direct-access storage occupied by that data set is erased (i.e., released for other use). The name is removed from the catalog if the data set was cataloged.

If the data set name designates a particular member of a partitioned data set, the member's name is deleted from the partitioned organization directory (POD) of that data set.

If the data set name is a partially qualified name or the name of a generation data group, all data sets (or generations) indexed under that dsname are erased and their catalog entries are removed.

If the name of a partitioned data set is supplied without a member name, the storage for the entire partitioned data set is erased, and its name is removed from the catalog.

PROGRAMMING NOTES: The ERASE macro instruction cannot be used to erase data sets on magnetic tape; it applies to data sets on direct-access storage only.

If a shared data set is opened by several users concurrently, a particular user cannot erase that data set until every other sharer actively using that data set issues a CLOSE macro instruction to deactivate their use of that data set. Any effort to erase an actively shared open data set will be ignored and result in diagnostics being issued. Once a user is the only currently active user of a shared data set he may erase that data set regardless of whether he has closed the data set or not.

If the data set name specifies SYSULIB (user library), it must also include the name of the module that is to be erased. The module name must be contained within parentheses following the SYSULIB data set name. Erasure of the entire SYSULIB data set is not permitted; therefore, specifying SYSULIB without a module name is invalid.

L- AND E-FORM USE: The L-form must specify the dsname as text (i.e., the data set name enclosed in apostrophes). The E-form must not specify the dsname, but must specify the location of the parameter list created by the L-form.

EXAMPLES: EX1 erases the data set A.B.C. EX2 erases all data sets cataloged under the partially qualified name A.B. EX3 erases the data set whose name is stored at location NAMLOC. EX4 removes member LAURA from the partitioned data set R.L.T. EX5 generates the parameter list for erasing data set M.P.S., and EX6 erases M.P.S.

```
EX1     ERASE    'A.B.C'
EX2     ERASE    'A.B'
EX3     ERASE    NAMLOC
EX4     ERASE    'R.L.T(LAURA)'
EX5     ERASE    'M.P.S',MF=L
EX6     ERASE    MF=(E,EX5)
```

REL -- Release Data Set or Remove Job Library From Program Library List (S)

The REL macro instruction deletes the definition previously established for a data set. It may be used, in effect, to cancel a preceding definition for either a public or private data set, as well as to release the input/output devices associated with a private data set. It may also be used to release one or all data sets of a given concatenation, and to remove JOBLIB from the user's program library list.

Name	Operation	Operand
[symbol]	REL	oplist-{text / addr}

oplist
specifies the list of operands.

Oplist
ddname-symbol [,dsname-name]

ddname
specifies a data definition name previously issued by a DDEF macro instruction or command. This data-definition name identifies the data set being released. The ddname may specify a job library and may also specify that the library data set name is to be removed from the program library list.

dsname
specifies the name of one data set in a concatenated series. If the operand is not specified, all data sets concatenated with the ddname will be released.

This operand can be specified as the fully qualified name of a non-partitioned data set or of a nonpartitioned generation of a generation data group (identified by absolute generation name or relative generation number).

CAUTION: When a data set has been released, it cannot be referred to again until another DDEF macro instruction or command defining that data set is issued.

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: After execution of the REL macro instruction, the low-order byte of register 15 contains one of these codes:

Code (Hexadecimal)	Significance
00	Normal
04	Defaulted or invalid ddname
08	Attention interrupt occurred
0C	Reserved ddname specified - not permitted
10	Undefined ddname
14	Uncataloged on public storage
18	Undefined dsname
20	Spurious input

L- AND E-FORM USE: The oplist operand is required in the L-form of this macro instruction and is not permitted in the E-form. Only the text form of the operand may be used in the L-form of the macro instruction.

EXAMPLES: In EX1, a character constant is given for the data-definition name DD1. In EX2, the address of the same ddname is given.

```
EX1     REL       'DD1'
EX2     REL       RELTAG
```

SECTION II: PROGRAM MANAGEMENT

This section describes TSS/360 macro instructions available to the user to facilitate program management. To enhance user understanding of these macro instructions, they are presented in functional groups that reflect their primary use in the system.

VIRTUAL STORAGE MANAGEMENT

It might occasionally become necessary for a user to obtain additional virtual storage space at some point during the execution of his program. The TSS/360 program management facilities provide a user with the services of several macro instructions (GETMAIN and FREEMAIN) to give him this capability. The need for additional virtual storage at object time is specified by issuing the GETMAIN macro instruction. When the dynamically allocated virtual storage area is no longer required by a user he may then release the area by issuing the FREEMAIN macro instruction.

In addition to acquiring additional storage space, the CSTORE macro instruction has been provided to enable a user to transform any set of contiguous virtual storage bytes into an object module, consisting of a single control section, during the execution of his program. These macro instructions are briefly described below.

GETMAIN allows a user to request a contiguous area of virtual storage be made available to him at some point during the execution of his program. The user can request a number of pages or bytes of such virtual storage be allocated to him. Subsequent processing within the user program can make use of the allocated area.

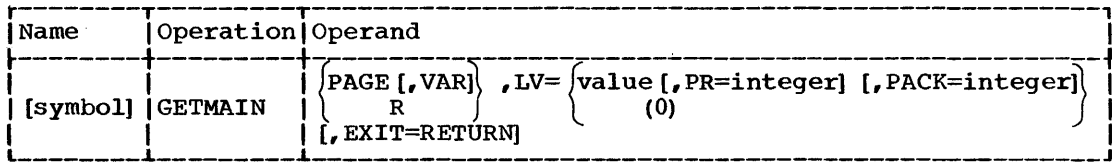
FREEMAIN releases a virtual storage area previously allocated by a GETMAIN macro instruction.

CSTORE transforms a number of contiguous virtual storage bytes from virtual storage to an object module consisting of one control section. The bytes of contiguous virtual storage will be released and returned to the virtual storage map for future allocation.

Detailed explanations of each of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to virtual storage management and its related macro instructions can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

GETMAIN -- Allocate Virtual Storage (R)

The GETMAIN macro instruction requests a contiguous area of virtual storage for a user's task at object time. The areas of virtual storage allocated by GETMAIN contain binary zeroes.



PAGE
specifies that a number of pages of virtual storage is to be allocated.

VAR
specifies a variable number of pages to be allocated, defined at system generation time by the installation. If the LV value is nonzero, these additional pages are added to the variable allocation.

R
specifies that a number of bytes of virtual storage is to be allocated.

LV
specifies the desired number of pages or bytes of virtual storage.

If (0) is written, the value must be given as a binary number placed in the low-order three bytes of register 0, right-adjusted, and the high-order byte of register 0 must contain binary 0.

The length of the specified virtual storage area may not exceed the amount of virtual storage available at the time of execution of the macro instruction. Refer to the EXIT operand.

PR
specifies the protection class to be assigned to the requested virtual storage. The following values may be specified:

0 - User read-and-write
1 - User read-only
2 - Private privileged

This parameter only has meaning for privileged users; if it is omitted, PR=0 is assumed. If bytes were specified and an invalid protection class is specified, a return code of 8 is placed in register 15.

PACK
specifies that the requested virtual storage be put into a unique segment or packed into the first available space as follows:

0 - put into a unique segment, or pack into the first available space, depending on system parameters and type of request
1 - pack into the first available space, regardless of any system parameters or the type of request
2 - put into a unique segment regardless of any system parameters or the type of request

This parameter may only be used by privileged users; if it is omitted PACK=0 is assumed.

EXIT=RETURN
specifies that, if the request for virtual storage cannot be satisfied, a return code of 4 is placed in general register 15.

CAUTION: If a request for virtual storage cannot be satisfied, and the EXIT operand is omitted, an abnormal task termination occurs.

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: Two sequential GETMAIN macro instructions do not guarantee the allocation of two contiguous areas. The only way to ensure a contiguous allocation of n pages is by issuing a GETMAIN macro instruction specifying an area whose length is n.

The address of the allocated virtual storage is returned in register 1. The area begins on a page boundary if pages of virtual memory were requested, and on a doubleword boundary if bytes were requested.

If the GETMAIN macro instruction is executed successfully, a return code of 0 is placed in register 15.

EXAMPLE:

```
EX1     GETMAIN   PAGE,LV=(0),EXIT=RETURN
EX2     GETMAIN   PAGE,LV=6
EX3     GETMAIN   PAGE,LV=(0)
```

EX1
specifies a request for pages and indicates that register 0 has been loaded with the number of pages of virtual storage requested, and with 0s in the high-order byte of the register. EX1 also specifies that a return code of 4 be issued if the request cannot be satisfied.

EX2
requests six pages of virtual storage to be allocated.

EX3
indicates a request for pages. Before execution of this macro instruction, the user loads register 0 with the length of the required area and loads zeros in the high-order byte of the register. If the virtual storage cannot be allocated, the task is abnormally terminated.

FREEMAIN -- Release Allocated Virtual Storage (R)

The FREEMAIN macro instruction releases a virtual storage area previously allocated by a GETMAIN macro instruction. This virtual storage area can be released by units of pages or 8 byte multiples.

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

PAGE
specifies that a number of pages of virtual storage is to be released.

VAR
specifies the release of an area of virtual storage obtained

through a PAGE,VAR GETMAIN macro instruction. The LV= parameter must be written as in the corresponding GETMAIN macro instruction.

R

specifies that a number of bytes of virtual storage is to be released.

LV

specifies the length, in pages or in bytes, of the virtual storage area, previously allocated by a GETMAIN macro instruction, to be released.

If (0) is written, the value must be given as a binary number placed in the low-order three bytes of register 0, right adjusted. The high-order byte of the register must be 0.

A

specifies the address of a fullword containing the address of the virtual storage area to be released.

If (1) is written, the address of the virtual storage area (not the address of a fullword containing the virtual storage area address) must be loaded into register 1 before execution of this macro instruction. If bytes are specified the address of the virtual storage area must be on a double word boundary or an error code of 8 is returned in register 15.

CAUTION: During execution of the FREEMAIN macro instruction, the task issuing the FREEMAIN macro instruction is abnormally terminated, if:

1. The area to be released is privileged or contains privileged areas, or
2. The area to be released was not allocated by a GETMAIN.

If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

If FREEMAIN is unable to locate the page or doubleword boundary, containing the virtual memory to be released, or if any of the virtual memory has never been assigned or has already been released, a return code of 4 is placed in register 15.

EXAMPLES:

```
EX1     FREEMAIN  PAGE,LV=16,A=(1)
EX2     FREEMAIN  PAGE,LV=(0),A=ADD1
EX3     FREEMAIN  PAGE,VAR,LV=2,A=ADD2
```

EX1

requests the release of a 16-page virtual storage area whose address is in register 1.

EX2

requests the release of an area whose address is in the fullword at ADD1 and whose length, in pages, is in register 0.

EX3

requests the release of an area, whose length is two pages more than the value specified at system generation time, and whose address is in the fullword at ADD2.

CSTORE -- Control Section Store (S)

The CSTORE macro instruction enables the user, during program execution, to transform any set of contiguous virtual storage bytes into an object module consisting of a single control section. The module is stowed in the current JOBLIB. It can then be loaded by the program that created it, or by a subsequent program. When the module is loaded, no relocation takes place; therefore, it may contain no relocatable items. The resulting module will consist of an unnamed control section which contains a copy of the hexadecimal text beginning at the page boundary corresponding to or preceding the address specified as the starting address parameter, and terminating at the page boundary corresponding toor following the address computed from the fourth parameter. Thus, the resulting control section will always be an integral number of pages in length.

When the module is loaded by the user, the module name, as well as the entry point name, will point to the address computed by adding to the load address of the new module, the page off-set (if any) implied by the starting address. For example, assume that the user requests that a control section of 4098 bytes be created from the bytes beginning at virtual storage address 5D050. Two pages of hexadecimal text beginning at the page boundary address (in this case 5D000) corresponding to or preceding the specified starting address will be transformed into an object module. The module and entry point names are off-set from the page boundary by 50 to reflect the actual address (5D050) of the hexadecimal text which the user desires to place in a control section. Assuming that the new module is later loaded at 70000, the loaded module and control section will occupy two full pages beginning at 70000. The second page is required so that the new control section will include the last two bytes requested by the user. The new module and entry point names will be adjusted to reflect the off-set and will both point to 70050.

Maximum control section size is one segment.

Name	Operation	Operand
[symbol]	CSTORE	module-symbol,epname-symbol,address-addr, length-value,attribute-value

module
: specifies the name to be assigned to the module created to contain the control section.

epname
: specifies the entry point name to be assigned to the specified address location.

address
: specifies the relocatable expression or register notation for the address of the first byte of data to be included in the control section.

length
: specifies, in bytes, the amount of data to be included in the control section.

attribute
specifies a control section attribute byte whose contents indicate:

Bit 0 on - System
Bit 1 on - Privileged
Bit 2 on - Common
Bit 3 on - Prototype (PSECT)
Bit 4 on - Public
Bit 5 on - Read-only
Bit 6 on - Variable length
Bit 6 off - Fixed-length

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: The control section is created from any contiguous set of bytes, and is an integral number of pages in length. A control section is not built if the module or entry-point names are duplicates of existing names in the current JOBLIB.

Subsequent loading of the created module is accomplished implicitly, by using an R and V type constant for the entry point name or module name, or explicitly, by use of the LOAD macro instruction.

The common attribute (bit 2), if specified, will be ignored by the dynamic loader, since it treats all unnamed control sections as unique. The created module may contain no relocatable words (adcons) and can be referenced by the control section name or module name offsets.

Upon completion of execution of the CSTORE macro instruction, the low-order byte of register 15 contains one of these return codes:

Code	Significance
00	normal return
04	Module name or entry point name already in use

L- AND E-FORM USE: The module and csname operands are required in the L-form and not permitted in the E-form of this macro instruction. All other operands are optional in the L- and E-forms; however, if an operand is omitted from the L-form, it must be specified in the E-form.

EXAMPLE: This example indicates the macro instruction used to create a module named MYMODULE which contains one unnamed control section. The control section consists of the two pages of text taken from the bytes beginning at HERE, which is on a page boundary. The entry point name EPNAME points to the beginning of the control section, which as public and read-only attributes.

```
EX1   CSTORE   MYMODULE,EPNAME,HERE,8000,12
```

DCLASS* -- Specify Privilege Class (O)

The DCLASS macro instruction declares the user as privileged or non-privileged and causes subsequent code to be executed accordingly.

RSPRV* -- Restore Privilege (O)

The RSPRV macro instruction responds to a nonprivileged routine's request to be returned to the privileged state.

CKCLS* -- Check Protection Class (O)

The CKCLS macro instruction determines the most restrictive protection class assigned to a specified number of contiguous half-pages.

LSCHP* -- List Changed Pages (R)

The LSCHP macro instruction lists changed pages of virtual storage for the user.

* Although these macro instructions are available to all users, they are employed primarily by system programmers; therefore, refer to System Programmer's Guide, Form C28-2008.

PROGRAM LOADING AND LINKING

A user has two ways of requesting that a module be loaded into virtual storage; either by an implied request or an explicit request.

An implied request causes a program to be automatically loaded by the system into a user's virtual storage during program assembly each time the source program references (i.e., via CALL macro instruction) an undefined external symbol. An explicit request is satisfied during the actual execution of the program containing the request. When the explicit request (specified via a CALL or LOAD macro instruction) is executed, the referenced module is loaded into virtual storage assigned to that task. Unlike the implicit call, the program loaded by an explicit call during program execution may be released by subsequent issuance of a DELETE macro instruction or an UNLOAD command. This would release the virtual storage area occupied by that program for further use by the user's program.

When a user's program calls another program, either explicitly or implicitly, these programs must establish linkage between one another using standard TSS/360 linkage conventions. Thus, proper registers must be used in establishing linkage, and a save area must be set aside in the calling program. Two macro instructions (SAVE and RETURN) have been provided to aid a user in establishing standard linkage.

ADCON generates a group of address constants which point to the program that is to be explicitly or implicitly loaded. The adcon group is generated in a format appropriate for the routine (i.e., the LOAD or CALL routine) for which they are being provided.

ADCOND generates a Dummy Control Section which provides predefined symbolic names for the fields of an explicit adcon group that was generated via an ADCON macro instruction. These names allow a user to symbolically access the resolved address constants that are placed in the explicit adcon group upon execution of a LOAD or explicit CALL macro instruction.

ARM initializes the ADCON group, defined by an ADCON macro instruction, with the name of the module, entry point, or control section that is to be loaded into virtual storage. The initialized adcon group can subsequently be used by a CALL or LOAD macro to explicitly load the indicated program into storage.

CALL causes the called program to be loaded into virtual storage either explicitly or implicitly and establishes conventional linkage between the calling and called program. The module or program to be loaded is located via the address constants previously defined by an ADCON or ARM macro instruction, which point to the called routine. This macro instruction also initiates execution of the called program.

LOAD causes a copy of a specified program to be loaded into the virtual storage of a task explicitly if it is not already there. The module of the program being loaded is located via the address constants previously defined by an explicit ADCON macro instruction. The program is placed in virtual storage and cannot be released from storage until the task logs off, a DELETE macro instruction is issued for that program, or an UNLOAD command is issued at the terminal. This macro instruction does not initiate the execution of the called program.

SAVE stores the contents of general registers in a save area provided in the program control section, or module which has called the routine in which the SAVE macro instruction is issued. The SAVE

macro instruction should normally be the first instruction in a called routine.

RETURN when issued in a called routine, it returns control to the calling routine at its NSI, restores general registers to their status at entry to the called routine, and passes return codes to the routine which had called it.

Detailed explanations of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to loading and linking can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

ADCON -- Generate an Adcon Group (O)

The ADCON macro instruction generates an adcon group for use by a CALL, a LOAD, or a DELETE macro instruction.

Name	Operation	Operand
symbol	ADCON	type-code [,EP=symbol] [,LDERR={CODE|ERR}] [,DELOPT={SMO|SDM}]

type
specifies the type of adcon group to be generated. Possible codes for the type operand, with their meanings are:

Code	Meaning
CALL	An explicit adcon group is generated for use by the CALL macro instruction.
LOAD	An explicit adcon group is generated for use by the LOAD macro instruction.
DELETE	An adcon group is generated for use by the DELETE macro instruction.
IMPLICIT	An implicit adcon group is generated using the externally defined symbol specified in the EP operand.
INTERNAL	An implicit adcon group is generated using the internally defined symbol specified in the EP operand.

EP
specifies the entry point of the module to which the adcon group refers. If EP is omitted, eight blank characters are used as the entry point name for CALL, LOAD, and DELETE adcon groups, whereas zero is used as the entry point address and R-value for IMPLICIT and INTERNAL adcon groups. Although the EP operand may be omitted from the ADCON macro instruction, the entry point must eventually be supplied to the appropriate field(s) of the adcon group before the latter is actually used. Refer to the ARM and ADCOND macro instruction descriptions in this section.

symbol
specifies the entry point of the module to which the adcon group

refers. For an INTERNAL adcon group, the R-value indicates the origin of the control section containing the ADCON macro instruction. An ADCON macro instruction that specifies INTERNAL as the type must consequently not be written in an unnamed control section.

LDERR
: specifies whether the Dynamic Loader is to take an error exit or to present a return code if the specified module cannot be loaded. If CODE is specified, ADCON generates bit ADCC2CB7 of the ADCC2C control byte with a value of 1 so that the Dynamic Loader will store in the same control byte a return code indicating the reason for loading failure. For a description of the return code values and their meanings, refer to the LOAD macro instruction in this publication. If ERR is specified or if LDERR operand is omitted, ADCON generates the bit described above with a value of 0 so that the Dynamic Loader will initiate "load error procedure" when the specified module cannot be loaded. The LDERR operand may be used only with a type operand that specifies LOAD or CALL.

DELOPT
: specifies the DELETE option desired. If SMO is specified, ADCON generates bit ADCC3DB7 of the ADCC3D control byte with a value of 1 so the Dynamic Loader will attempt to delete only the specified module. If SDM is specified or if the DELOPT operand is omitted, ADCON generates the bit just described with a value of 0 so the Dynamic Loader will attempt to delete all modules on which the specified module depends, as well as the specified module itself. The DELOPT operand may be used only with a type operand that specifies DELETE.

PROGRAMMING NOTES: An explicit adcon group is altered the first time a CALL or LOAD macro instruction refers to it. In this altered state, the adcon group is said to be disarmed; before being altered, it is said to be armed. The ADCON macro instruction may be used to generate a fully armed explicit adcon group having all control bytes and the entry point name generated with the desired values. The user may, however, want to complete arming by supplying the entry point name or control byte settings after the adcon group is generated. In any case, an explicit adcon group must be fully armed the first time it is used by a LOAD or loadtype-E CALL macro instruction.

Once an adcon group has been disarmed during loading or calling of a program, it may subsequently be used in that state only for one purpose and under certain conditions. If the program that was loaded or called has not been deleted, and if the adcon group used in its loading or calling has not been modified either by the ARM macro instruction or by the user's own code, the same adcon group may be used in subsequent calls to the same program. A disarmed adcon group may be made available for the following purposes only if it is rearmed by means of the ARM macro instruction:

- Calling or loading the same program again after it has been deleted.
- Calling the same program again after the referenced adcon group has been rearmed for a different program.
- Calling or loading a different program.

Note that an explicit adcon group generated for use by the LOAD macro instruction must not be used by the CALL macro instruction and vice versa, except in the following situation. An explicit adcon group that is used to load a program may, if it is not subsequently modified eitherby ARM or by the user's own code, and if the loaded program is not subsequently deleted, be used in subsequent calls to the loaded program.

If the user issues ADCON macro instructions, the V-con and R-con pair are located at a displacement of 12 from the label used for the ADCON macro instruction.

The user may refer directly to certain fields of adcon groups of any type. These fields are described below; no other fields can ever be altered directly by the user. The name for each field or bit position is the name provided by the ADCOND macro instruction. All references to adcon group fields and bit positions must use these names.

EXPLICIT ADCON GROUPS FOR USE WITH LOAD OR CALL MACRO INSTRUCTIONS:

Field Name		
For LOAD	For CALL	Meaning
ADCC1L	ADCC1C	Control byte 1
ADCC1LB7	ADCC1CB7	Bit of control byte 1; specifies type of explicit adcon group; bit is 0 for LOAD; 1 for CALL adcon groups
ADCC2L	ADCC2C	Control byte 2
ADCC2LB7	ADCC2CB7	Bit of control byte 2; corresponds to the LDERR operand
ADCPNAM	ADCPNAM	Eight-byte field containing as a character constant the name of program to be loaded or called

DELETE ADCON GROUP

Field Name	Meaning
ADCC3D	Control byte 3
ADCC3DB7	Bit of control byte 3; corresponds to DELOPT operand
ADCC4D	Control byte 4 in which Dynamic Loader places return code indicating results of DELETE request; return code of 0 indicates successful deletion; code X'04' indicates no deletion took place because module defining specified EP symbol was not present in user's virtual storage when request for deletion was given; code X'08' indicates no deletion took place because of other outstanding references to specified program
ADCPNAMD	Eight-byte field containing as a character constant the name of program to be deleted

IMPLICIT ADCON GROUPS

Field Name	Meaning
ADCEP	A four-byte adcon, aligned on a fullword boundary, containing entry point of specified program (V-value)
ADCRV	A four-byte adcon, aligned on a fullword boundary, containing R-value of specified program

CAUTION: Because adcon groups must be capable of being changed, they must not be generated in read-only control sections.

EXAMPLE 1: This coding sequence generates an implicit adcon group for calling EXNAM, an externally defined entry point name:

```
         LA       15,LEXNAM
         CALL     (15),,,E
         .
         .
         .
LEXNAM   ADCON    IMPLICIT,EP=EXNAM
```

EXAMPLE 2: This coding sequence generates a DELETE adcon group for deleting only EXNAM, the specified module. EXNAM is assumed to have been previously loaded.

```
         .
         .
         .
         DELETE   EPLOC=LEXNAM
         .
         .
         .
LEXNAM   ADCON    DELETE,EP=EXNAM,DELOPT=SMO
```

ADCOND -- Provide Symbolic Names for an Explicit Adcon Group (O)

The ADCOND macro instruction generates a dummy control section (DSECT) that provides symbolic names for the fields in an explicit adcon group. The name of the generated DSECT is CHAADC.

This DSECT permits symbolic access to the resolved V-type and R-type address constants which are placed in the explicit adcon group upon execution of a LOAD or explicit CALL macro instruction. The control byte C_2, which directs the loader to a course of action, may also be accessed. For an explanation of the control byte, refer to the LOAD macro instruction in this section.

Name	Operation	Operand
	ADCOND	

CAUTION: The ADCOND macro instruction may be used only once in an assembly.

PROGRAMMING NOTES: The C_1 control byte is addressable by the following names: ADCC1C (for CALL) or ADCC1L (for LOAD). The C_2 control byte is addressable as ADCC2C (for CALL) or ADCC2L (for LOAD). The C_3 control byte for DELETE is addressable by the symbolic name ADCC3D; the C control byte for DELETE has the symbolic name ADCC4D. The symbolic nameADCVCON addresses the resolved V-type address constant. The symbolic name ADCRCON addresses the resolved R-type address constant. When ADCC1C is set to '00' (hexadecimal) a LOAD explicit adcon group is implied, and when set to '01' (hexadecimal) an explicit CALL adcon group is implied.

The macro instruction may appear at any point in a control section. However, if it is written at any location other than at the end of a control section, the original control section must be resumed.

EXAMPLE: The following example illustrates how a program accesses a field in an explicit adcon group. The program alters the C_2 control byte so that the loader will return codes which indicate the action of the loader. Refer to the LOAD macro instruction in this section.

The ADCON macro instruction generates an explicit adcon group for a LOAD. ARM readies the adcon group for use by a LOAD. The LA instruction places the address of the adcon group into register 5. A USING statement establishes a base register for CHAADC. The MVI instruction sets the C_2 control byte to 1; this setting requests the loader to return codes when the adcon group is used by a LOAD.

```
            .
            .
            .
RALPH   ADCON    LOAD
            .
            .
            .
        ARM      RALPH,SQROUT
        LA       5,RALPH
        USING    CHAADC,5
        MVI      ADCC2L,X'01'
            .
            .
            .
            .
SQROUT  DC       CL8'SQROUT'
        ADCOND
```

ARM -- Initialize an Explicit Adcon Group (O)

The ARM macro instruction initializes an explicit adcon group, so that it may be used by a loadtype-E CALL macro instruction or a LOAD macro instruction.

Explicit adcon groups must be initialized if:

1. They are to be used for the first time to refer to one program where they have been used at least once for a different program.
2. Adcon group has been used at least once and the associated program has been deleted by the DELETE macro instruction.
3. They were generated by an ADCON macro instruction without the EP operand.

Name	Operation	Operand
[symbol]	ARM	loc-addrx,extref-addrx

loc
specifies the address of adcon group to be initialized.

extref
specifies the address of an eight-byte field that contains the external name that is to be placed in the explicit adcon group; i.e., the name of the module, entry point, or control section to be loaded or called.

PROGRAMMING NOTES: After execution of the ARM macro instruction, register 15 contains the address of the armed adcon group.

CALL -- Call a Module (S)

The CALL macro instruction passes control from one module to another module or from one point in a module to another point within the same module.

The module issuing the CALL macro instruction is referred to as the calling module; the module receiving control is referred to as the called module.

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

entry
specifies the symbolic name of an entry point to which control is to be passed. If the module is not reenterable, the symbolic name can be: the name of a control section; the name in the operand field of an assembler language ENTRY statement; or a module name. If the module is reenterable, control section name must not be used. If (15) is written, and the loadtype is I, the address of an implicit adcon group must be loaded into register 15 before execution of this macro instruction. If (15) is written and the loadtype is E, the address of an explicit adcon group must be loaded into register 15 before execution of this macro instruction.

An implicit adcon group consists of two contiguous fullwords: the V-type and R-type address constants of the entry point. These address constants must be coded as a V-type followed by an R-type. See Example 2, below.

The explicit adcon group may be generated through the ADCON macro instruction. The ARM macro instruction can be used to reinitialize the adcon group.

```
L       ADCON     CALL, EP=entry point name
```

Refer to the ADCON and ARM macro instructions in this section.

param
specifies addresses to be passed as a parameter list to the called program. The param operands must be written as a sublist, as shown in the format description. If one or more param operands are written, a parameter list is generated. It consists of a fullword for each operand. Each fullword is aligned on a fullword boundary and contains 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 contains the address of the parameter list.

VL
specifies that the first word preceding the parameter list contains a binary number equal to the number of parameters (including null parameters) supplied by the param operand.

The operand parameter list is fixed-length if it contains a known number of parameters every time the called program is given control. The list is variable length if it contains a varying number

of parameters. In the latter case, the VL operand should be written so that the called program can determine the length of the parameter list being passed to it.

loadtype
specifies an explicit CALL, if E is written; an implicit CALL, if I is written. The default condition is I.

ID
specifies a binary calling sequence identifier for the CALL macro instruction. The maximum value is 4095. This parameter may be used to uniquely identify the CALL macro instruction. This parameter generates a NOP in TSS/360.

PROGRAMMING NOTES: The explicit CALL macro instruction causes the named module to be loaded (if necessary) at object time; it may then be deleted through use of the DELETE macro instruction. Refer to the DELETE macro instruction in this section. If an implicit CALL macro instruction is issued, the called object module is already in virtual storage and may not be deleted by the calling object module through use of the DELETE macro instruction.

If (15) is written for the entry operand of a loadtype E CALL macro instruction, the explicit adcon group should be armed if necessary and then reused for any subsequent CALLs to the desired program. ADCON is capable of generating an armed adcon group; refer to the ARM macro instruction in this section. However, the explicit adcon group is altered by the execution of the first CALL macro instruction and cannot be reused if the module has been deleted (refer to the DELETE macro instruction). If an object module has not been loaded or has been loaded and then deleted and it is desired to CALL it using a previously used explicit adcon group, it is necessary to issue or reissue the ARM macro instruction. The ARM macro instruction adjusts the explicit adcongroup so that it may be reused. Refer to examples 4 and 5, the ADCON macro instruction, and the ARM macro instruction.

If the entry operand is given as an internal symbol, it must appear as the operand of an assembler language ENTRY statement. The reason for this rule is that the called name must be in the program module dictionary (PMD), if the CALL macro instruction is to execute properly.

Register 14 contains a valid return address when control is passed to the called module. Therefore, by issuing a RETURN macro instruction or branching to the address in register 14, control is transferred to the instruction after the CALL macro instruction in the calling module. The CALL macro instruction is advantageous because it eliminates the need for writing linkage to the called module.

L- AND E-FORM USE: The L- and E-forms of this macro instruction are written as described in "S-Type Macro Instructions" in Section I except for the following special operand requirements:

Operand	L-Form	E-Form
entry	ignored	required
param	allowed	allowed[1]
VL	allowed[2]	allowed[2]
load-type	ignored	allowed
ID	ignored	allowed

[1]E-form param list entries overlay the corresponding L-form param list entries.
[2]If VL is specified on the E-form, it must have been specified on the L-form; if VL is not specified on the E-form, it must not have been specified on the L-form.

This example shows L- and E-form use:

```
ALPHA   CALL  ,(A,,C),MF=L
BETA    CALL  RTNA,(,B,),ID=36,MF=(E,ALPHA)
```

EXAMPLES: Typical implicit and explicit CALL use.

EXAMPLE 1 - Implicit CALL:

```
EX1     CALL  ENT
```

When the CALL macro instruction in the calling program is executed, control is passed to ENT.

EXAMPLE 2 - Implicit adcon group for an implicit CALL:

```
EX2     CALL  (15),(ABC,DEF),VL
```

Calling program contains an implicit adcon group:

```
SAMNAM  ADCON IMPLICIT,EP=CLDRTN
```

Before the CALL macro instruction is executed, register 15 must be loaded with the address of the adcon group; e.g., LA 15,SAMNAM.

When the called program is entered, register 1 points to a two-word parameter list. The first word contains the address of ABC; the second word, the address of DEF. The word preceding the parameter list contains a 2, indicating that two words containing the addresses of parameters follow.

EXAMPLE 3 - Explicit CALL:

```
EX3     CALL  ATOL,(BAT,CAT),,E
```

At execution time, the program whose entry point name is ATOL is loaded into virtual storage (if necessary) and control is transferred to ATOL. When the called program is entered, register 1 points to a two-word parameter list which contains the addresses of BAT and CAT. Register 14 contains the return address.

EXAMPLE 4 -- Repetitive Explicit CALLs, reusing an explicit adcon group:

```
              .
              .
              .
               ARM      MAX,JOE
              CALL      (15),,,E
                .
                .
                .
              CALL      (15),,,E
                .
                .
                .
              CALL      (15),,,E
                .
                .
                .
     MAX      ADCON     CALL
     JOE       DC       CL8'CALLEE'
```

EXAMPLE 5 -- Repetitive Explicit CALLs with an intervening DELETE and reusing an explicit adcon group.

```
              .
              .
              ARM       MAX,JOE
              CALL      (15),,,E
                .
                .
                .
              DELETE    EP=CALLEE
                .
                .
                .
              ARM       MAX,JOE
              CALL      (15),,,E
                .
                .
                .
     MAX      ADCON     CALL
     JOE       DC       CL8'CALLEE'
```

LOAD -- Load and Retain a Module (R)

The LOAD macro instruction is used to load a specified object module, and all other object modules to which it (and they) are explicitly linked, into the user's virtual storage. LOAD does not initiate program execution. The specified module cannot be released until the task logs off, executes a DELETE macro instruction, or executes an UNLOAD command. (Refer to Command System User's Guide.) Note that this macro instruction does not initiate execution of the specified program.

Name	Operation	Operand
[symbol]	LOAD	{EP=symbol EPLOC={addrx (1)}

EP
specifies the symbolic name of an entry point in the module to be loaded. The name must be the name of a control section, the name in the operand field of an assembler language ENTRY statement, or a module name.

EPLOC
specifies the address of the explicit adcon group representing the module to be loaded. If (1) is written, the address of the explicit adcon group must be loaded into parameter register 1 before execution of this macro instruction.

This adcon group can be generated by:

```
L          ADCON     LOAD,EP=external name
```

PROGRAMMING NOTES: If the module has already been loaded, this macro instruction is ignored.

The ADDC2L byte may be used to direct the dynamic loader to a course of action when the specified module cannot be loaded. If ADDC2L is set to 0, the loader takes a system prescribed error exit.

If an explicit adcon group is to be used for a LOAD macro instruction, it must first be initialized unless it has not yet been used and it was generated by an ADCON macro instruction. (Refer to the ARM and ADCON macro-instructions in this section.) If a loaded module has been deleted and it is desired to load it again, the same explicit adcon group may be reused provided it is reinitialized. After the LOAD macro instruction has completed, register 15 contains the address of the specified entry point in the loaded module.

EXAMPLE 1:

If a module whose entry point name is ROGER, is to be loaded, the following ADCON macro instruction is specified:

```
TERI       ADCON     LOAD,EP=ROGER
```

Upon execution, LOAD EPLOC=TERI causes the module associated with the entry-point name ROGER to be placed into virtual storage.

EXAMPLE 2:

```
LOAD EP=ALPHA
```

Upon execution of the LOAD macro instruction, a copy of the module associated with the entry-point name ALPHA is placed into virtual storage.

EXAMPLE 3:

```
LOAD EPLOC=(1)
```

Before issuing the LOAD macro instruction, the user loads the address of TERI into register 1; e.g., LA 1,TERI. The effect of this instruction is then the same as in Example 1.

DELETE -- Delete a Loaded Module (R)

The DELETE macro instruction indicates that a copy of a specified module, which was placed in virtual storage, is no longer required. This specified module must have been previously acquired by the issuan-

ceof a LOAD macro instruction or an explicit CALL macro instruction. Upon execution of this macro instruction, the specified module is deleted from the issuing task's virtual storage.

Name	Operation	Operand
[symbol]	DELETE	{EP=symbol EPLOC={addrx (1)}

EP
specifies the external name of the module to be deleted. This external name must be the name of a control section, the name in the operand field of an assembler language ENTRY statement, or a module name.

EPLOC
specifies the address of the delete adcon group representing the module to be deleted. If (1) is written, the address of the adcon group must have been loaded into parameter register 1 before execution of this macro instruction.

This delete adcon group is generated by:

```
ADCON     DELETE,EP=external name
```

EXAMPLE 1:

If the module associated with the external name EARL is to be deleted, and the following ADCON macro instruction is supplied:

```
DAVE    ADCON     DELETE,EP=EARL
```

The macro instruction MAX DELETE EPLOC=DAVE causes the module associated with EARL to be deleted.

EXAMPLE 2:

```
SARP    DELETE    EP=ALPHA
```

The module associated with the external symbol ALPHA is deleted.

EXAMPLE 3:

```
NAM     DELETE    EPLOC=(1)
```

Before this DELETE macro instruction is executed, the address of the delete adcon group must be loaded into register 1; e.g., LA 1,FAM. The effect of this macro instruction is then the same as in Example 1.

SAVE -- Save Register Contents (O)

The SAVE macro instruction is normally written at each entry point of a called program. Upon entry to the program, SAVE stores the contents of specified registers in a save area provided by the calling program. The saved register contents may then be restored by a RETURN macro instruction, assembler language LM instruction, or other programming techniques.

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

reg_1,reg_2
specify the range of registers whose contents are to be stored in the save area defined by the calling program that is pointed to by register 13. The operands are written as decimal numbers so that, when inserted in an assembler language STM instruction, they cause the contents of the desired registers in the range of 14 through 12 (i.e., 14, 15, and 0 through 12) to be stored. The contents of register 14 and 15, if specified, are saved in words 4 and 5 of the save area; the contents of registers 0 through 12, if specified, in words 6 through 18. The contents of a given register are always saved in a particular word in the save area. For example, the contents of register 3 are always saved in word 9 of the save area, even if contents of register 2 are not saved.
The reg_1 and reg_2 operands must not request saving the contents of register 13 which is the pointer to the save area.

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

Note: T and id are parameters used to facilitate tracing; i.e., checking program flow. There is no tracing in Time Sharing System/360; these parameters are provided for compatibility with the IBM System/360 Operating System.

T
specifies that the contents of registers 14 and 15 are to be saved in words 4 and 5 of the save area, if not already saved by the reg_1, reg_2 operands. If the T and reg_2 operands are present and the reg_1 operand is 14, 15, 0, 1, or 2, the contents of 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 of as many as 255 characters; it must contain no blanks or commas. Because it can have a length greater than eight characters, the operand 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 this macro instruction. If the name field is blank, the name of the control section containing the SAVE macro instruction is used.

PROGRAMMING NOTES: If the called routine is to use register 13, it must save the contents of register 13 and, before termination, restore it. The SAVE macro instruction must not be used for this saving.

When the macro is expanded, both the entry-point identifier and the count of the number of bytes in the identifier, in that order, are placed in front of the actual entry point to the SAVE routine. The entry point identifier is assembled starting at the nearest possible halfword boundary preceding the actual entry point. Because the count byte always immediately precedes the entry point an extra byte is sometimes needed to achieve the required halfword alignment for the identifier string. When the extra byte is needed, a character blank is

inserted at the end of the entry point identifier, immediately preceding the count byte. The count byte will contain a count equal to the number of characters in the identifier plus the blank (if used). The count byte, itself, is not included in the count.

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 assembly module is to branch to the entry point, the entry-point name should be an operand of an assembler-language ENTRY statement provided by the user in the current assembly module.

EXAMPLES: EX1 saves the contents of registers 14 through 10. The contents of registers 1 and 15 (and registers 0 and 1) 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.

Examples	Macro Expansions
EX1 SAVE (2,10),T,F4RTNA7B99	DC 0H DC CL11'F4RTNA7B99',FL1'11'
	EX1 STM 14,10,12(13)
EX2 SAVE (3,4),,*	DC 0H DC CL3'EX2',FL1'3'
	EX2 STM 3,4,32(13)

RETURN -- Return to a Program (O)

The RETURN macro instruction, issued in a called program, returns control to the calling program. The function of this macro instruction depends on how it is used.

1. If the first program to receive control from the system issues a RETURN macro instruction to return control to the system, the effect is the same as if an EXIT macro instruction had been issued.

2. A program which follows Type I linkage conventions and is given control by the CALL macro instruction, can return control to the program which called it by issuing a RETURN macro instruction.

The RETURN macro instruction may also be used to restore the contents of the registers of the calling program that were saved by the SAVE macro instruction issued in the called program.

Name	Operation	Operand
[symbol]	RETURN	[(reg$_1$-integer[,reg$_2$-integer])] [,T] [,RC={absexp / (15)}]

reg$_1$, reg$_2$
specifies the range of registers whose contents are to be restored from the save area of the calling program. The operands are written as decimal numbers such that, when inserted in an LM instruc-

tion, they cause the contents of the desired registers in the range from 14 through 12 (i.e., 14, 15, and 0 through 12) to be restored.

The contents of registers 14 and 15, if specified, are restored from words 4 and 5 of the save area; the contents of registers 0 through 12, if specified, restored from words 6 through 18. If reg_2 is omitted, only the register content specified by reg_1 is restored. If both reg_1 and reg_2 are omitted, no register contents are restored.

The address of the save area defined by the calling program must be loaded into register 13 before execution of this macro instruction.

T

specifies that a (1) is set in the low-order bit of the forward link, word 3, in the save area defined by the calling program. This action occurs after completion of the register reloading specified by the first operand. The bit is set to stop the forward chain.

This parameter is supplied to facilitate tracing; i.e., checking program flow. There is no tracing in TSS/360; this parameter is provided for compatibility with the IBM System/360 Operating System.

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 must be a multiple of 4 in the range from 0 through 4092 or diagnostics will be issued.

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

PROGRAMMING NOTES: The contents of 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 reg_1 and reg_2 operands must not specify that the contents of register 13, the save area register, are to be restored. If the contents of register 13 are to be saved and restored, it should be done according to linkage conventions as described in Appendix E. The RETURN macro instruction assumes that save area register (register 13) is correctly positioned to the save area defined by the calling program.

If no return code is specified, the contents of the return code register (register 15) will not be changed unless reg_1 and reg_2 span the return code register. When a RETURN macro instruction terminates a program, the return code in register 15 can be interrogated by the calling program.

EXAMPLES: In the following examples, EX1 is a RETURN macro instruction that restores the contents of registers 2 through 10. A 1 is placed in the low-order bit of word 3 in the save area. EX2 restores the contents of registers 14 through 5 and places a return code of 12 in register 15. EX3 does not restore the contents of any register; however, control is returned to the calling program.

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

DELET* -- Enter DELETE Service Routine (O)

The DELET macro instruction passes control to the task monitor for the explicit purpose of entering the DELETE service routine.

DLINK* -- Dynamic Linkage Request (O)

The DLINK macro instruction passes control to the task monitor for the explicit purpose of entering the dynamic loader.

ENTER* -- Enter a Privileged Routine (O)

The ENTER macro instruction enables a privileged routine to be entered from a nonprivileged routine.

INVOKE* -- Transfer Control (O)

The INVOKE macro instruction uses the restricted linkage conventions to transfer control from one program or routine to another.

LIBESRCH* -- Locate Program Module in External Library (S)

The LIBESRCH macro instruction determines if a particular program module has been defined in any of the libraries opened for the current task (i.e., LOGON to LOGOFF) and what library the module is located on.

RESUME* -- Restore Registers (O)

The RESUME macro instruction restores the contents of the specified registers from the area specified and unconditionally returns control to the calling program.

STORE* -- Store Register Contents (O)

The STORE macro instruction stores the contents of the specified register or registers in a specified area.

* Although these macro instructions are available to all users, they are employed primarily by system programmers; therefore, refer to System Programmer's Guide, Form C28-2008, for a discussion of these macro instructions.

INTERRUPT HANDLING FACILITIES

TSS/360 provides interrupt handling facilities which permit the user to control task interrupts. User coded routines can be invoked to service interrupts; these routines must decide how to respond to each type of interrupt and can even elect to ignore certain interrupts. The macro instructions which are available to a user for creating interrupt handling routines are briefly summarized below.

SIR defines a user routine (named via a SPEC, SAEC, SIEC, SEEC, STEC, or SSEC macro instruction) to the system as an interrupt handling routine for a specific type of interrupt and specifies the processing priority for that routine. This routine replaces any system supplied interrupt handling routines permanently, for this type of interrupt, unless it is subsequently deleted from the system via the DIR macro instruction. If DIR is issued, the system supplied interrupt handling routines are reinstated.

SPEC names a user coded program interrupt servicing routine and provides system control reference areas in which data pertaining to a program interrupt can be recorded. The named routine must be defined to the system as an interrupt handling routine by a SIR macro instruction.

SSEC names a user coded supervisor call (SVC) interrupt servicing routine and provides system control reference areas in which data pertaining to an SVC interrupt can be recorded. The named routine must be defined to the system as an interrupt handling routine by a SIR macro instruction.

SEEC names a user coded external interrupt handling routine and provides system control reference areas in which data pertaining to an external interrupt can be recorded. The named routine must be defined to the system as an interrupt handling routine by a SIR macro instruction.

SAEC names a user coded asynchronous interrupt handling routine and provides system control reference areas in which data pertaining to asynchronous interrupts can be recorded. The named routine must be defined to the system as an interrupt handling routine by a SIR macro instruction.

STEC names a user coded timer interrupt handling routine and provides system control reference areas in which data pertaining to timer interrupts can be recorded. The named routine must be defined to the system as an interrupt handling routine by a SIR macro instruction.

SIEC names a user coded input/output interrupt handling routine and provides system control reference areas in which data pertaining to I/O error interrupts can be recorded. The named routine must be defined to the system as an interrupt handling routine by a SIR macro instruction.

DIR deletes a user coded routine, previously defined to the system as an interrupt handling routine by a SIR macro instruction, from the system, thereby eliminating its use as an interrupt handling routine. The routine can no longer service interrupts unless redefined to the system as an interrupt handling routine be a SIR macro instruction.

SAI saves the task's current interrupt servicing status indicator and specifies that interrupts are not to be allowed during subsequent execution in that program. Interrupts are inhibited until a sub-

sequent RAE macro instruction is issued. Interrupts occurring while the inhibit indicator is on are saved and queued for handling when the indicator is reset to enable.

RAE restores the tasks interrupt servicing status previously saved by a SAI macro instruction and depending on what that status is (i.e., enable or inhibit) continues processing. If the previous status was enable, any interrupts that occurred while the program was in the inhibit state, are then processed.

INTINQ generally used in an interrupt servicing routine to examine the interrupt information recorded in the control reference areas established by a SIR macro instruction for the interrupt routine in which INTINQ is issued.

USATT causes subsequent attention interrupts to be processed by a user coded routine that was previously established as an interrupt handling routine by the SIR and SAEC macro instructions.

CLATT causes subsequent attention interrupts to be processed by the system; it relinquishes a user's control of attention handling interrupts previously granted by a USATT macro instruction.

AETD causes subsequent attention interrupts to be processed by any one of several user coded routines depending on the number of times the attention key is hit. Control of attention interrupts is returned to the system when an AETD macro instruction with no operand is issued.

Detailed explanation of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to interrupt handling and the related macro instructions can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

SIR -- Specify Interrupt Routine (S)

The SIR macro instruction establishes control references for the user's interruption routine and specifies its processing priority. The control references inform the system of the presence of interrupt control blocks that specify the interruptions to be serviced by user routines.

Name	Operation	Operand
[symbol]	SIR	(icb-addr,...) [,PRTY=integer] [,INHIBIT=YES|NO]

icb
specifies the address of an interrupt control block (ICB) established by a SPEC, SAEC, SIEC, SSEC, STEC or SEEC macro instruction.

INHIBIT
specifies whether the interruption routines established by this macro instruction may be interrupted by a higher priority interruption routine. If this operand is invalid or omitted, the option YES is assumed for privileged programs and NO for nonprivileged programs where:

YES
specifies that the routine may not be interrupted by higher priority routines.

NO
specifies that the routine may be interrupted by higher priority routines.

PRTY
is a decimal integer specifying the processing priority for the interruption routine defined by the ICB referred to by the icb operand. Priorities from 0 to 127 are available for nonprivileged routines, and 128-240 are reserved for privileged routines. An assembly error message will be provided if the priority is invalid. If the priority is invalid or omitted, a priority of 0 is assumed for nonprivileged routines and a priority of 128 for privileged routines.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: The priority of an interruption routine is actually determined by two factors: First, is the priority specified in its SIR macro instruction. Priority increases with the magnitude of the priority code; e.g., priority 10 is higher than 9. The second factor pertains to two or more interruption routines specified with the same priority; e.g., for the same device (for I/O and asynchronous ICBs) or same interruption type (for SVC, Timer, Program and External). Within such a group, priority increases with the order in which the routines were specified; the highest priority goes to the routine appearing last in the last SIR macro instruction.

The priority of an interruption routine may be changed by another SIR macro instruction, but only if the control references have first been deleted by a DIR macro instruction.

If the same ICB is referred to more than once in a single SIR macro instruction, only the first reference is used; subsequent references are ignored.

Normally, re-issuing a SIR on an ICB that has been previously SIRRED will result in an immediate return with an error code of '08' in register 15. However, if an ICB created via a STEC macro instruction has a second SIR issued for it without any intervening DIR macros being issued, the timing intervals that were previously set in the ICB are automatically reset to their original values. In this case, the user must re-SIR the same ICB with the same priority and the same inhibit option, in order to get proper results.

At object time, the following error conditions cause immediate return, with the error code in register 15, and the address of the invalid ICB in register 1:

Code	Condition
04	ICB contains invalid DCB (for input/output and asynchronous ICBs only) or invalid time or clock number specified (for timer).
08	ICB specified previously by another SIR macro instruction.
0C	Invalid parameter (ICB or length invalid, or a nonprivileged user requests privileged priority or has privileged flag on in parameter area).
10	Total user time will exceed 71/2hours. Time not set.

L- AND E-FORM USE: The parameter list (list of ICB addresses) set up by L-form of this macro instruction can also be used by the DIR macro instruction, for deletion of interruption routines.

An ICB address of 0 in the L-form of the macro instruction will cause space to be reserved in the list. If this address is still 0 when the E-form is executed, it is treated as a no-operation (NOP).

EXAMPLE 1:

```
LIST1   SIR       (0,0,C,D),PRTY=5,MF=L
```

This macro instruction will, when executed, construct a list of four ICB addresses, two of which are 0, and will define a priority of 5 for each ICB.

EXAMPLE 2:

```
EX2     SIR       (,B),MF=(E,LIST1)
```

When executed, this macro instruction places the address of B (an ICB) into the second ICB address field of LIST1, and then establishes control references for each ICB in the list, ignoring the first because the address is still 0.

EXAMPLE 3:

```
EX3     SIR       (A,0,0,0),MF=(E,LIST1)
```

When executed, this macro instruction places the address of A into the first ICB address field of LIST1, sets the remaining three fields to 0, and then establishes control references to A.

SPEC -- Specify Program Entry Conditions (S)

The SPEC macro instruction creates an interrupt control block (ICB) to service program interrupts, and specifies the address of a communication area and the interrupt handling routine's entry point.

Name	Operation	Operand
[symbol]	SPEC	[EP=symbol] [,COMAREA=addr] [,INTTYP={((A|S|R),{inttype-{code|integer|integer-integer}},...) ((,inttype-{code|integer|integer-integer},...){NULL|SAVE|RESTORE}}] [,MF=L|(E,icb-{addrx|(1)})]

EP
specifies the entry point name of the interrupt routine to which control is to be transferred when an interruption of the type specified by INTTYP occurs.

COMAREA
specifies the address of an area in main storage, aligned on a fullword boundary, that is to be used by the control program to pass interruption information to the interruption routine.

INTTYP
specifies the types of interruptions that will cause entry to the interruption routine; where

A
specifies that the interruption information (code) that follows is to be added to the existing INTTYP field of the ICB.

S
specifies that the interruption information (code) that follows is to be subtracted from the existing INTTYP field of the ICB.

R
specifies that the interruption information (code) that follows is to replace the existing INTTYP field of the ICB.

integer
consists of one or more decimal integers (1 through 15), representing program interruptions. A range of interruptions can be indicated by two integers separated by a hyphen; e.g., 7-11.

code
specifies a mnemonic for one of the 15 program interruptions. The integer values and code mnemonics are:

Integer	Code	Meaning
1	OP	operation
2	M	privileged
3	EX	execution
4	P	protection
5	AD	addressing
6	SP	specification
7	D	data
8	IF	fixed-point overflow
9	IK	fixed-point divide
10	DF	decimal overflow
11	DK	decimal divide
12	E	exponent overflow
13	U	exponent underflow
14	LS	significance
15	FK	floating-point divide

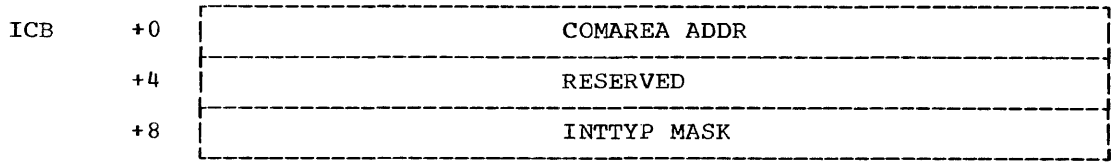
NULL
indicates that none of the types of interruptions covered by INTTYP is to be serviced.

SAVE
specifies that the contents of the INTTYP field of the ICB are to be saved. If this or NULL or RESTORE is written, the A, S, or R and interruption type codes are not written.

RESTORE
specifies that the contents of the INTTYP field of the ICB are to be replaced with the mask saved by an INTTYP=SAVE operand in a previous SPEC macro instruction.

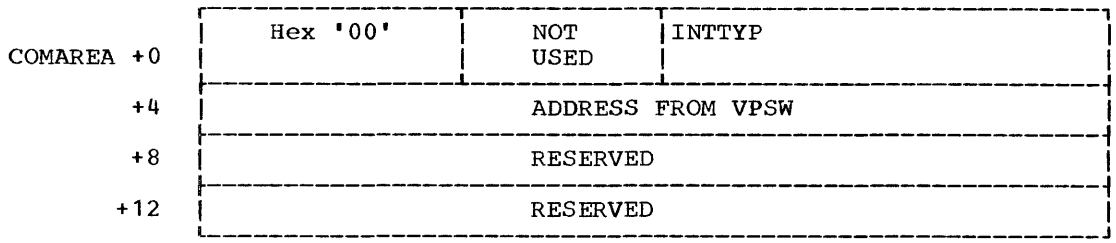
icb
specifies the address of the interrupt control block.

PROGRAMMING NOTES: The format of the first three words of the interrupt control block is:



INTTYP MASK is arranged so that bit 1=operation, bit 2=privileged operation, etc. (See INTTYP.)

Upon entry to an interruption routine, the COMAREA will contain the information relating to the interruption to be serviced. The format of the communication area and a description of its contents are shown here.



INTTYP
is a hexadecimal value from 1 to 15 representing the program interrruption type.

L- AND E-FORM USE: If neither L- nor E-form is specified, L is assumed. There is no standard S-type function for this macro instruction since no linkage is performed. The in-line code for the E-form is to alter the contents of an ICB. Therefore the operand MF=(E,icb-addr) with no other operands is meaningless and produces an assembly error message.

The A, S, R, SAVE or RESTORE operands are written only in the E-form of the macro instruction.

EXAMPLE 1:

```
EX1     SPEC    EP=PROG1,COMAREA=COM1,INTTYP=(1,3,5-10),MF=L
        DS      0F
COM1    DS      CL16
```

The ICB may be referred to by the symbolic name EX1. Conditions are defined for an interruption routine whose initial entry point is the location specified by the symbolic name PROG1. Only program interruption types 1, 3 and 5-10 will be processed by this interruption routine. If one of those interruptions occurs, an entry will be made to PROG1 and the interruption data will be in the first two words of COM1 and the address of EX1 will be in register 1. The first word of EX1 will contain the address of COM1.

EXAMPLE 2:

```
EX2     SPEC    INTTYP=(A,11-15),MF=(E,EX1)
```

This macro instruction will, when executed, cause the ICB EX1 to be modified to include program interruptions 11 to 15 to be processed by the routine with entry point PROG1.

Macro Instruction EX2 could also be written as:

```
EX2     SPEC     INTTYP=(A,DK,E,U,LS,FK),MF=(E,EX1)
```

Note that the macro instruction EX2 could be within the interruption routine PROG1.

SEEC -- Specify External Entry Conditions (S)

The SEEC macro instruction creates an interrupt control block (ICB) to service external interruptions, and specifies the address of a communication area and the interrupt handling routine's entry point.

Note: This macro instruction should be used only by those programs which may also use the VSEND macro instruction.

Name	Operation	Operand
[symbol]	SEEC	[EP={symbol / (0)}] [,COMAREA=addr] [,INTTYP=integer] [,MSGAREA=addr], [,MSGLTH=integer] [,MF=L|(E,icb-{addrx / (1)})]

EP
: specifies the entry point of the interruption routine to which control is to be transferred when an interruption occurs for the message specified by INTTYP.

  If this operand is omitted or if EP=0 is specified, the user must place the entry point name of the interruption routine in the parameter list generated by this macro instruction.

COMAREA
: specifies the address of an area in main storage, aligned on a fullword boundary, that is to be used by the control program to pass interruption information to the interruption routine.

INTTYP
: specifies the message number (0-240) that will cause entry to the interrupt routine. Message numbers 0 to 127 are reserved for privileged programs, and message numbers 219 to 240 are reserved for nonprivileged programs. Numbers 128 to 218 are not available for use.

MSGAREA
: specifies the address of an area into which the message is to be moved.

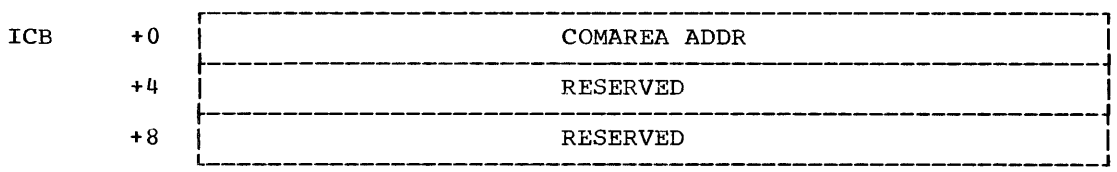
MSGLTH
: specifies the length in doublewords of the message. This operand may not specify a value greater than 238.

ICB
: specifies the address of the interrupt control block.

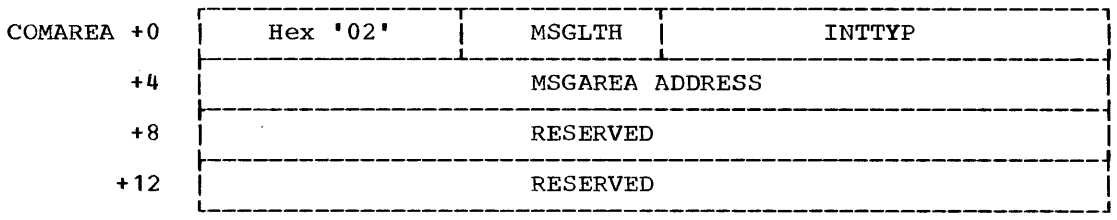
CAUTION: If an interruption routine is to serve multiple messages, a separate ICB must be defined (SEEC macro instruction) and specified (SIR macro instruction) for each message, and the routine must be reenterable.

PROGRAMMING NOTES: The message number (INTTYP) in an ICB should not be changed while the associated routine is active (currently processing or interrupted before completion of its processing) without first deleting the interruption routine with a DIR macro instruction. After changing the INTTYP, the routine must be re-established with a SIR macro instruction.

The format of the first three words of the interrupt control block is:



Upon entry to an interruption routine, the COMAREA will contain the information relating to the interruption to be serviced. The format of the communication area and a description of its contents are shown here.



MSGLTH
specifies the length (0-238) in doublewords of the message which has been moved.

INTTYP
specifies the message number (0-255) that will cause entry to the interruption routine. Message numbers 0 to 127 are reserved for nonprivileged programs, and message numbers 219 to 255 are reserved for privileged programs. Numbers 128 to 218 are not available for use.

MSGAREA
specifies the starting address of an area into which the message has been moved.

L- AND E-FORM USE: If neither L- nor E-form is specified, L is assumed. There is no standard S-type function for this macro instruction since no linkage is performed. The in-line code for the E-form is to alter the contents of an ICB. Therefore the operand MF=(E,icb-addrx) with no other operands is meaningless and will produce an assembly error message.

EXAMPLE 1:

```
ICBE1   SEEC    EP=PROG1,INTTYP=4,COMAREA=AREA1,MSGAREA=AREA2,
                MSGLTH=72,MF=L
        DS      0F
AREA1   DS      CL16
AREA2   DS      72D
```

The ICB may be referred to by the symbolic name ICBE1. Conditions are defined for an interruption routine whose initial entry point is the location specified by the symbolic name PROG1. When an interruption

formessage #4 (as specified by the INTTYP operand) causes entry to PROG1, the interruption data will be present in the first four words of AREA1, and the address of a parameter list containing the address of the COMAREA will be in register 1.

EXAMPLE 2:

```
SEEC    INTTYP=6,MF=(E,ICBE1)
```

This macro instruction will, when executed, cause the ICB defined in example 1 to be modified, allowing interruptions for message #6 to be processed by the routine with entry point at PROG1.

SSEC -- Specify Supervisor Call Entry Conditions (S)

The SSEC macro instruction creates an interrupt control block (ICB), to service SVC interrupts, and specifies the address of a communication area and the interrupt handling routine's entry point.

Name	Operation	Operand
[symbol]	SSEC	[EP={symbol / 0}] [,COMAREA=addr] [,INTTYP=integer] [,MF=L | (E,icb-{addrx / (1)})]

EP
: specifies the entry point of the interruption routine to which control is to be transferred when an interruption of the type specified by INTTYP occurs.

  If this operand is omitted or if EP=0 is specified, the user must place the entry point name of the interruption routine in the parameter list generated by this macro instruction.

COMAREA
: specifies the address of an area in main storage, aligned on a fullword boundary, that is to be used by the control program to pass interruption information to the interruption routine.

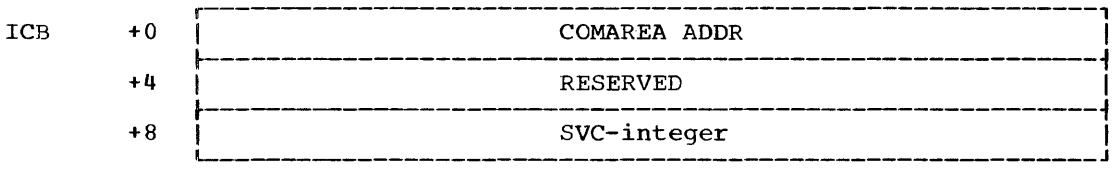
INTTYP
: specifies the I field of the SVC instruction the execution of which will cause entry to the interruption routine. It may have a value from 0 to 63.

icb
: specifies the address of the interruption control block.

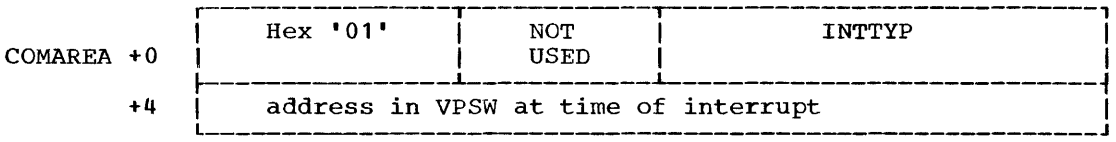
CAUTION: If an interruption routine is to serve multiple SVCs, a separate ICB must be defined (SSEC macro instruction) and specified (SIR macro instruction) for each SVC to be serviced and the routine must be reenterable.

PROGRAMMING NOTES: The INTTYP in an ICB should not be changed while the associated routine is active (currently processing or interrupted before completion of its processing) without first deleting the interruption routine with a DIR macro instruction. After changing the SVC-integer, the routine must be reestablished with a SIR macro instruction.

The format of the first three words of the interrupt control block is:



Upon entry to an interruption routine the COMAREA will contain the information relating to the interruption to be serviced. The format of the communication area and a description of its contents are:



INTTYP
identifies the SVC I field value 0 to 63.

L- AND E-FORM USE: If neither L- nor E-form is specified, L is assumed. There is no standard S-type function for this macro instruction since no linkage is performed. The in-line code for the E-form is to alter the contents of an ICB. Therefore the operand MF=(E,icb-addrx) with no other operands is meaningless and will produce an assembly error message.

EXAMPLE 1:

```
EX1     SSEC    EP=SVC1,COMAREA=COM1,INTTYP=12,MF=L
        DS      0F
COM1    DS      CL20
```

The ICB may be referred to by the symbolic name EX1. Conditions are defined for an interruption routine whose initial entry point is the location specified by the symbolic expression SVC1. An interruption caused by the execution of an SVC with an I field of 12 will cause entry to SVC1. Register 1 will contain the address of the ICB, the first word of which contains the address of COM1.

EXAMPLE 2:

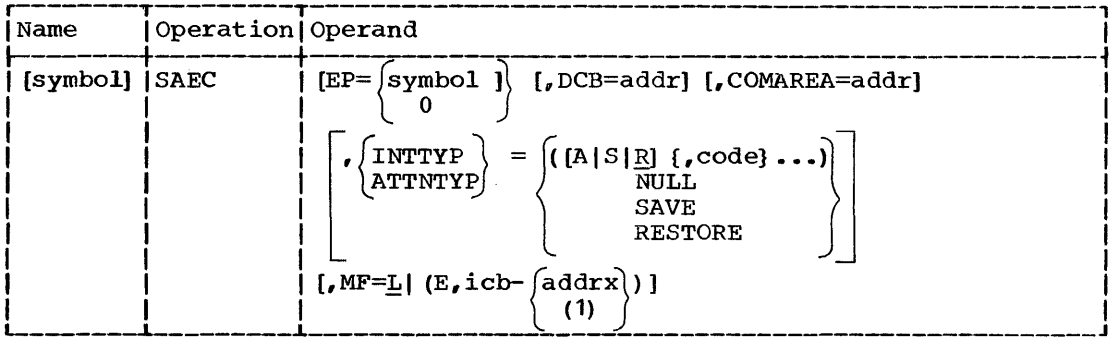
```
EX2     SSEC    INTTYP=29,MF=(E,EX1)
```

This macro instruction will, when executed, cause the interrupt control block EX1 to be modified to change the SVC I field handled by the SVC1 interruption routine from 12 to 29.

Note: An ICB has not been defined to the system until it has been specified in a SIR macro instruction.

SAEC -- Specify Asynchronous Entry Conditions (S)

The SAEC macro instruction creates an interrupt control block (ICB) to service asynchronous interruptions, and specifies the address of a communication area, the data control block, and the interrupt handling routine's entry point.



EP
: specifies the entry point of the interruption routine to which control is to be transferred when an interruption of the type specified by INTTYP occurs.

  If this operand is omitted or if EP=0 is specified, the user must place the entry point name of the interruption routine in the parameter list generated by this macro instruction.

DCB
: specifies the address of a previously opened data control block associated with the unit for which the routine is to service interruptions.

COMAREA
: specifies the address of an area in main storage, aligned on a fullword boundary, that is to be used by the control program to pass interruption information to the interruption routine.

INTTYP or ATTNTYP
: specifies types of interruptions that will cause entry to the interruption routine.

  A
  : specifies that the interruption information (code) is to be added to the existing INTTYP field of the ICB.

  S
  : specifies that the interruption information (code) is to be subtracted from the existing INTTYP field of the ICB.

  R
  : specifies that the interruption information (code) is to replace the existing INTTYP field of the ICB.

  code
  : specifies the type or types of interruptions to be added to, subtracted from, or to replace the INTTYP field of the ICB, and can be written as one or more of the following: ATTN, CANCEL, ALL, EOS, and AE. These codes are:

    ATTN
    : indicates an attention interruption.

    CANCEL
    : indicates that the routine is to service interruptions from the CANCEL key on the alphameric keyboard. The CANCEL key should be reserved to request control program intervention.

ALL
indicates that the routine is to service interruptions from all sources.

EOS
indicates that the routine is to service interruptions caused by execution of end-of-order-sequence orders.

AE
indicates that the routine is to service interruptions caused by asynchronous errors.

NULL
indicates that none of the types of interruptions covered by INTTYP are to be serviced.

SAVE
specifies that the contents of the INTTYP field of the icb are to be saved. If this or NULL or RESTORE is written, the A, S, or R and interruption type codes are not written.

RESTORE
specifies that the contents of the INTTYP field of the ICB are to be replaced with the mask saved by an INTTYP=SAVE operand in a previous SAEC macro instruction.

icb
specifies the address of the interrupt control block.

CAUTION: If an interruption routine is to serve multiple units, a separate ICB must be defined (SAEC macro instruction) and specified (SIR macro instruction) for each unit and the routine must be reenterable.

PROGRAMMING NOTES: The data control block address in an ICB should not be changed while the associated routine is active (currently processing or interrupted before completion of its processing) without first deleting the interruption routine with a DIR macro instruction. After changing the DCB address, the routine must be re-established with a SIR macro instruction.

The format of the first three words of the interrupt control block is:

ICB	+0	COMAREA ADDR
	+4	DCB ADDR
	+8	RESERVED

Upon entry to an interruption routine, the COMAREA will contain the information relating to the interruption to be serviced. The format of the communication area and a description of its contents are:

COMAREA	+0	Hex '03'	RESERVED	INTTYP
	+4	SENSE DATA		
	+8	RESERVED		
	+12	RESERVED		

INTTYP
indicates the type of interruption that occurred, by one of the following hexadecimal codes:

Code	Type
04	EOS (end-order-sequence)
05	CANCEL key (or ATTN)
06	AE (asynchronous error)

L- AND E-FORM USE: If neither L- nor E-form is specified, L is assumed. There is no standard S-type function for this macro instruction since no linkage is performed. The in-line code for the E-form is to alter the contents of an ICB. Therefore the operand MF=(E,icb-addrx) with no other operands is meaningless and will produce an assembly error message.

The A, S, R, SAVE, and RESTORE operands are specified only in the E-form of this macro instruction.

EXAMPLE 1:

```
ICBX1   SAEC    EP=AR1,DCB=GRAPHD1,INTTYP=(ALL),
                COMAREA=AREA1,MF=L
        DS      0F
AREA1   DS      CL16
```

The ICB may be referred to by the symbolic name ICBX1. Conditions are defined for an interruption routine whose initial entry point is the location specified by the symbolic name AR1. All interruptions will be processed by this interruption routine. When an interruption on the device specified by the DCB operand GRAPHD1 causes entry to AR1, the interruption data will be present in the first four words of AREA1; the address of a two-word parameter list containing the addresses of the COMAREA and the DCB will be in register 1.

STEC -- Specify Timer Entry Conditions (S)

The STEC macro instruction creates an interruption control block (ICB) to service timer interruptions, and specifies the address of a communication area and the interruption handling routine's entry point.

Name	Operation	Operand
[symbol]	STEC	[EP={symbol / (0)}] [,COMAREA=addr] [INTTYP={TASKn ,{DINTVL=addr / BINTVL=addr} / REALn ,{DINTVL=addr / BINTVL=addr / TOD=addr / DO{W|M|Y}=addr}}] [,MF=L| (E,icb-{addrx / (1)})]

EP
specifies the entry point name of the interruption routine to which control is to be transferred when an interruption of the type specified by INTTYP occurs.

If this operand is omitted or if EP=0 is specified, the user must place the entry point name of the interruption routine in the parameter list generated by this macro instruction.

COMAREA
specifies the address of an area in main storage, aligned on a fullword boundary, and at least 16 bytes long, that is to be used by the control program to pass interrupt information to the interruption routine.

INTTYP
specifies the types of interruptions that will cause entry to the interruption routine; where:

TASK
specifies that the interval is to be decremented only when the task issuing the STEC macro instruction is in control. A user may not request an interval timer to be set with a total task time that would be greater than 7.5 hours.

REAL
specifies that the interval is to be decremented continuously whether or not the task issuing the STEC macro instruction is in control.

n
specifies one of the 16 programmed timers. It must be an integer.

0 to 7 for nonprivileged programs
8 to 15 for privileged programs

DINTVL
specifies the address of a doubleword containing a decimal interval to be set into the timer. If real clock time was indicated (REAL), the doubleword must be aligned on a doubleword boundary and contain eight unpacked, unsigned decimal digits in the format HHMMSSth, where HH ≤ 23, MM ≤ 59, SS ≤ 59, t ≤ 9, h ≤ 9. If task time is being set (TASK), the maximum time interval that may be specified is 7.5 hours.

BINTVL
specifies the address of a fullword containing a binary interval to be established for this task. If real clock time was indicated (REAL), the fullword must be aligned on a fullword boundary and contain a positive 32-bit binary number in which the least significant bit has a value of 0.001 second. The specified interval must be less than 24 hours. If task time is being set (TASK), the maximum time interval that may be specified is 7.5 hours.

TOD
specifies the address of a doubleword containing the time of day at which the interval is to end. The doubleword must be aligned on a doubleword boundary and contain eight unpacked decimal digits in the format HHMMSSth (defined in the DINTVL operand).

DOW
specifies the address of a four-byte field containing the day of the week at which the interval is to end. The four bytes must contain one of the following seven character combinations.

MOND
TUES
WEDN
THUR

FRID
SATU
SUND

DOM
specifies the address of a two-byte field containing the day of the month at which the interval is to end. The byte contains two unpacked decimal digits which must take on a value in the range 01 through 31.

DOY
specifies the address of a fullword containing five packed decimal digits of the form YYDDD, where YY = the last two digits of the year and DDD = the day of the year. These five digits are preceded by two packed decimal zeros and followed by a four-bit character such that all digits will have the same zone if the 32-bit word is unpacked.

icb
specifies the address of the interruption control block.

Note: The TOD, DOW, DOM and DOY operands are meaningful only when a REAL interval is specified. If a TASK interval is specified an error message will be issued at assembly time.

PROGRAMMING NOTES: Timers set via the STEC macro instruction are decremented as specified and must be reset before each entry to the timer interrupt servicing routine. These timers can be reset simply by issuing a second SIR macro instruction without any intervening DIR macro having been issued.

The format of the first three words of the interrupt control block is:

ICB	+0	COMAREA ADDR
	+4	RESERVED
	+8	RESERVED

Upon entry to an interruption routine, the COMAREA will contain the information relating to the interruption to be serviced. The format of the communication area and a description of its contents are:

COMAREA	+0	Hex '04'	T/R	INTTYP
	+4	RESERVED		
	+8	RESERVED		
	+12	RESERVED		

T/R
specifies the character T (TASK) or R (REAL) identifying the type of timer which has caused the interruption.

INTTYP
indentifies the timer (0-15) that has caused entry into the interruption routine.

L- AND E-FORM USE: If neither L- nor E-form is specified, L is assumed. There is no standard S-type function for this macro instruction since no linkage is performed. The in-line code for the E-form is to alter the contents of an ICB. Therefore the operand MF=(E, icb-addrx) with no other operands is meaningless and will produce an assembly error message.

EXAMPLE:

```
     EX1     STEC      EP=TIME1,COMAREA=COM1,INTTYP=TASK5,DINTVL=TADDR1,
                       MF=L
             DS        0F
     COM1    DS        CL16
```

SIEC -- Specify Input/Output Entry Conditions (S)

The SIEC macro instruction creates an interrupt control block (ICB) to service I/O interruptions, and specifies the addresses of a communication area, a data control block, and the interrupt handling routine's entry point.

Name	Operation	Operand
[symbol]	SIEC	[EP={symbol / 0}][,DCB=addr] [,COMAREA=addr] [,MF=L | (E,icb-{addrx / (1)})]

EP
: specifies the entry point name of the interrupt routine to which control is to be transferred when a synchronous interruption for the unit specified in the DCB occurs.

  If this operand is omitted or if EP=0 is specified, the user must place the entry point name of the interruption routine in the parameter list generated by this macro instruction.

DCB
: specifies the address of a previously opened data control block associated with the unit for which the routine is to service interruptions.

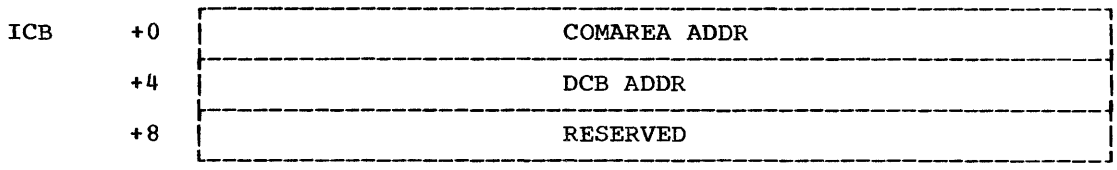
COMAREA
: specifies the address of an area in main storage, aligned on a fullword boundary, that is to be used by the control program to pass interruption information to the interruption routine.

icb
: specifies the address of the interrupt control block.

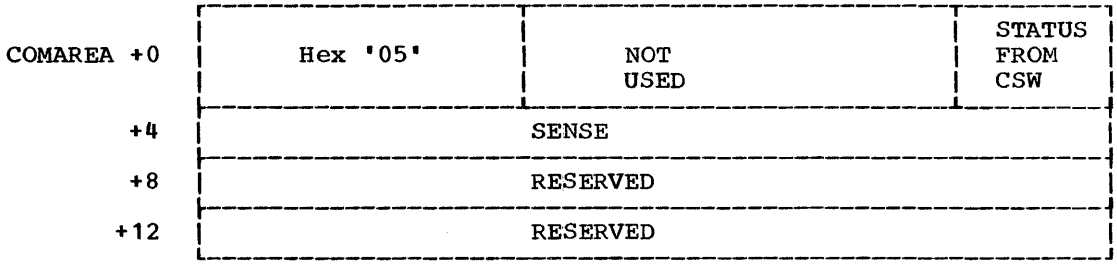
CAUTION: If an interruption routine is to service multiple units, a separate ICB must be defined (SIEC macro instruction) and specified (SIR macro instruction) for each unit, and the routine must be reenterable.

PROGRAMMING NOTES: The DCB address in an ICB should not be changed while the associated routine is active (currently processing or interrupted before completion of its processing) without first deleting the interruption routine with a DIR macro instruction. After changing the DCB address, the routine must be reestablished with a SIR macro instruction.

The format of the first three words of the interrupt control block is:



Upon entry to an interruption routine, the COMAREA will contain the information relating to the interrupt to be serviced. The format of the communication area is:



L- AND E-FORM USE: If neither L- nor E-form is specified, L is assumed. There is no standard S-type function for this macro instruction since no linkage is performed. The in-line code for the E-form is to alter the contents of an ICB. Therefore the operand MF=(E,icb-addr) with no other operands is meaningless and will produce an assembly error message.

EXAMPLE 1:

```
EX1     SIEC    EP=PROG1,COMAREA=COM1,DCB=DCB1,MF=L
        DS      0F
COM1    DS      CL16
```

The ICB may be referred to by the symbolic name EX1. Conditions are defined for an interruption routine whose initial entry point is the location specified by the symbolic name PROG1. If an interruption for the unit specified in DCB1 occurs, an entry will be made to PROG1 and the interruption data will be in the first two words of COM1. The address of EX1 will be in register 1. The first word of EX1 will contain the address of COM1.

EXAMPLE 2:

```
EX2     SIEC    DCB=DCB2,MF=(E,EX1)
```

This macro instruction, when executed, causes the ICB EX1 to be modified to handle I/O interruptions for the unit specified in DCB2.

Note that the macro instruction EX2 might be within the interrupt routine PROG1.

DIR -- Delete Interrupt Routine (S)

The Delete Interrupt Routine (DIR) macro instruction deletes control references to a previously specified interrupt control block. The interruption routine specified in the ICB cannot service interruptions unless the ICB is respecified by a SIR macro instruction.

Name	Operation	Operand
[symbol]	DIR	(icb-addr,...)

icb
specifies the address of an interrupt control block established by a SPEC, SAEC, SIEC, SSEC, STEC, or SEEC macro instruction.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: A DIR macro instruction deletes an active routine (one currently processing or interrupted) or prevents a routine from receiving subsequent interruptions through use of an E-form SPEC or SAEC macro instruction, using the NULL code for the INTTYP operand.

At execution time, the following conditions cause a return, with a return code in register 15, and the address of the invalid ICB in register 1. These registers indicate the last return condition if multiple return conditions are encountered.

Return Code	Condition
04	ICB contains invalid DCB (for input/output and asynchronous ICBs only) or an invalid time interval or clock number was specified (for timer).
08	No routine specified.
0C	the interrupt servicing routine is active (no further interrupts will be presented to the interrupt routine until it has completed its current servicing action).
10	invalid parameter (an invalid length was specified or a nonprivileged user has attempted to DIR a privileged routine).

L- AND E-FORM USE: The list operand in the E-form of the macro instruction may refer to the same list of ICBs used by the SIR macro instruction.

SAI -- Save and Inhibit (O)

The SAI macro instruction saves the inhibit status of the task monitor and sets the problem program in the inhibit state.

Name	Operation	Operand
[symbol]	SAI	[area-addr]

area
specifies a one-byte area for saving the prior inhibit status of the task monitor. If this operand is omitted, only the inhibit function occurs.

PROGRAMMING NOTES: The task monitor always dispatches in the inhibit state to privileged routines unless the interruption program was made available to the system by a SIR macro instruction with the operand INHIBIT=NO.

The task monitor dispatches in the enabled state to nonprivileged interruption programs unless the interruption program was made available to the system by a SIR macro instruction with the operand INHIBIT=YES.

There are separate inhibit indicators for privileged and nonprivileged programs. The SAI macro instruction sets the appropriate indicator dependent on the attributes of the program being assembled. A nonprivileged program cannot inhibit the dispatching to privileged programs.

RAE -- Restore and Enable (O)

The RAE macro instruction restores the prior inhibit state of the task monitor and sets the problem program in the enabled state.

Name	Operation	Operand
[symbol]	RAE	[area-addr]

area
: specifies a one-byte area previously used by an SAI macro instruction for saving the prior task monitor inhibit status. If this operand is omitted, the restore function is not executed but the enable status is set.

PROGRAMMING NOTES: There are separate enable indicators for privileged and nonprivileged programs. The RAE macro instruction sets the appropriate indicator depending on the attributes of the program being assembled. A nonprivileged program cannot inhibit dispatching to privileged programs.

INTINQ -- Interrupt Inquiry (O)

The INTINQ macro instruction relinquishes control until more information is available, maintains control in a wait state, or sets up a conditional branch. It causes an examination of interruption queued information for an ICB defined as available to the system by a SIR macro instruction.

Name	Operation	Operand
[symbol]	INTINQ	icb-addr [,MODE={R / W / CLEAR / (C,branch-addr),TYP=code}]

icb
: specifies the address of an ICB which has been defined available to the system by means of a SIR macro instruction. This ICB should not be one which has been defined to the system with a lower

priority than the ICB by which current entry to this routine was made if both include the address of the same data control block.

MODE
specifies one of four modes of inquiry and can be specified as R, W, CLEAR, or C. If C is specified, the "branch" and TYP operands are required. If the MODE operand is not written, the R (relinquish) option is assumed.

R
specifies that the interruption routine relinquishes control until more interruption information of the type specified in the ICB associated with the interruption routine is available. If this information has already been queued by the system, this routine may immediately regain control. Control returns to this routine at the instruction following the INTINQ macro instruction.

W
specifies that the interruption routine enters a wait condition pending availability of interruption information of the type specified by the icb operand. Control is not relinquished although the wait condition may be interrupted by a routine of higher priority. At the end of the wait, execution is resumed with the instruction following the INTINQ macro instruction.

CLEAR
specifies deletion of any interruptions queued for the routine indicated by the ICB operand. Processing continues with the next sequential instruction.

Note: These queued interruptions may or may not conform with interrupt types currently defined in the ICB.

C
specifies a branch to be taken to the location specified by the "branch" operand, if the information specified by the TYP operand is found in the queue of interruption information. If it is not found, execution is resumed with the next sequential instruction. The "branch" operand must be written if the C option is chosen.

branch
specifies the address to which control is to be transferred if interruption information of the type specified by the TYP operand is available.

TYP
specifies the type of interruption information to be the condition for the branch. The code can be written as any of the INTTYP codes (as described in the SPEC, SAEC, SSEC, STEC and SEEC macro instructions in this section) as long as the INTTYP is consistent with the type of ICB defined by the ICB operand in this macro instruction. ANY is written if any interruption information of the type specified in the associated ICB is desired. TYP can be something other than the INTTYP specified in the associated ICB. TYP associated with SIEC should specify ANY.

CAUTION: If this module is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: The INTINQ macro instruction inspects the queue of interruption information; the subsequent course of action is determined by the availability of interruption information and the mode specified in the macro instruction. Determination of the subsequent action for modes R, W, and C is illustrated in the following chart.

Mode	Required Interrupt Information: Available	Required Interrupt Information: Not Available	Action
R	X		Continue execution with next sequential instruction
		X	Relinquish control; resume execution with next sequential instruction when information available
W	X		Continue execution with next sequential instruction
		X	Enter wait state until information available; then continue with next sequential instruction
C	X		Branch to specified branch address
		X	Continue execution with next sequential instruction
The INTINQ macro instruction may be issued only from an interruption routine. Standard register conventions are required with the INTINQ macro instruction; register 13 must contain the address of a save area for this interrupt routine.			

Conditions that cause special return codes are listed below, with the associated hexadecimal return code. These conditions apply only to the modes stated.

04 Undefined routine specified (modes C, W, CLEAR)

08 Bad parameter list, the conditions specified can never be met (modes W, C)

USATT -- Give User Control of Attention Interrupts (O)

The USATT macro instruction allows the user to have his own routine process attention interruptions.

Name	Operation	Operand
[symbol]	USATT	

PROGRAMMING NOTES: The user must first issue the SAEC and SIR macro instructions to establish the routine that is to process attention interruptions. He then issues the USATT macro instruction, and all subsequent attention interruptions will be processed by the specified routine. However, if no routine has been established, the user loses control of his task.

Once the user gains control of attention interruptions by issuing a USATT macro instruction, control can be returned to the system by using either the CLATT, EXIT, CLIC, CLIP, PAUSE, or COMMAND macro instructions. If the user program issues the CLIC, CLIP, PAUSE, or COMMAND macro instructions, the system regains control of attention interrupts until a RUN command (without an operand) is issued. When a CLATT or EXIT macro instruction gives control of attention interrupts to the system, issuing a RUN command does not automatically return control of interrupts to the user. In this case he can only regain control by issuing another USATT macro instruction in this program.

In the SAEC macro instruction used to set up user control of attention interrupts, the DCB parameter must be specified as SYSINDCB.

CLATT -- Give System Control of Attention Interrupts (O)

The CLATT macro instruction allows the user to relinquish control of attention interruptions; the system then processes attention interruptions.

Name	Operation	Operand
[symbol]	CLATT	

PROGRAMMING NOTES: This macro instruction is used in conjunction with the USATT macro instruction, discussed in this section.

AETD -- Create an Attention Entry Table (O)

The AETD macro instruction allows a user to interrupt his programs during execution by hitting the attention key, and thereby enter a predefined user coded subroutine to process the attention interrupt.

Name	Operation	Operand
[symbol]	AETD	[({[ep-symbol,sa-symbol]},...)]

ep
: specifies the symbolic entry point name of a routine to be entered upon hitting the attention key at the terminal.

sa
: specifies the symbolic name of a 21-word save area that is to be associated with the routine whose entry point is ep. The 21-word save area is provided in addition to the standard 19-word save area (which must be provided in order to conform to standard linkage conventions). The two additional words in the 21 word save area, are for saving the VPSW.

PROGRAMMING NOTES: The execution of the AETD macro instruction generates a table containing the addresses of routines which are to be given control when a user hits the attention key a specified number of times (i.e., a user may specify that different routines be entered when the attention key is hit a varying number of times). Thus, with the first hit of the attention key, the user's program execution is inter-

rupted and Procedure 1 in the table would be initiated; if he hits the attention key a second time before Procedure 1 has been completed, he will enter Procedure 2, and if he hits the attention key a third time before Procedure 2 has been completed, he will enter Procedure 3, and so on for as many predefined procedures as desired. Procedures specified in this manner are generally used to communicate with the user's terminal allowing program modification at execution time.

The user might employ the AETD macro instruction to pass control to any user provided control systems, or to provide partial backup in a current task so that a bad error situation does not have to cause the task to be reconstructed from scratch. It can be used to predefine simple automatic debugging procedures by using PCS commands in the AETD attention handling routines.

The table generated (Attention Entry Table, AET) consists of three words containing V - and R -con and save area addresses for each attention handling routine that a user has specified. Any null operand pair causes three words containing binary zero to be created in the table. Entries are generated in the same order as given in the operands.

If AETD is issued with no operand, the current table (AET) , if one was previously defined, will be disconnected from the system and the system attention handling routines will be invoked for subsequent processing of attention interrupts.

If the save area or entry point is externally defined, it must be used as an argument of an EXTRN statement in the user's program.

An A - type and R- type adcon pair is normally generated for each entry point name; in this case, the R - value is the origin of the first declared PSECT in the assembly module containing AETD. If an entry point is externally defined, AETD generates a V-type and R-type adcon pair for that entry point operand. An A-type adcon is also normally generated for each save area address.

L- AND E- FORM USE: The L- and E-forms of this macro instruction are allowed and have no special requirements. The E-form of the macro instruction may specify any parameters; however, the parameters specified in the E-form will overlay those specified in the L-form. The E-form may not specify more operands than are specified in the corresponding L-form.

For example:

```
SUE     AETD  (ETRYPTA,SAVEA) ,MF=L
        AETD  (,SAVEB) ,MF= (E,SUE)
```

When the E-form of this macro instruction is executed, the save area specified in the L-form (SAVEA) will be replaced in the parameter list by the save area specified in the E-form (SAVEB)

EXAMPLE: In the following example, the user has provided two attention handling routines having the entry points EPMODA and EPMODB respectively. If the user hits the attention key at the terminal once following execution of the first AETD macro instruction, control will be passed to the user coded routine at EPMODA. When the user hits the attention key a second time before the routine at EPMODA has completed execution, control will be immediately passed to the routine at EPMODB. Execution of a second AETD macro instruction, having no operand, will return control of attention interrupts to the appropriate system routines.

```
        .
        .
AETD  (EPMODA,SAVA,EPMODB,SAVB)
```

.
.
AETD
.

ATPOL* -- Poll For Pending Attention Interrupt (nonstandard)

The ATPOL macro instruction determines if there are any pending attention (task-asynchronous) asynchronous I/O interrupts; if there is, control is transferred to a routine specified by the user.

ITI* -- Inhibit Task Interrupts (nonstandard)

The ITI macro instruction prevents the occurrence of task interrupts until a subsequent PTI macro instruction is executed.

PTI* -- Permit Task Interrupts (nonstandard)

The PTI macro instruction allows any pending task interrupts to occur; it is used to cancel the effect of a previously issued ITI macro instruction.

PCSVC* -- Enter Program Checkout Subsystem (nonstandard)

The PCSVC macro instruction creates a task-SVC that transfers control to the task monitor or program checkout subsystem (PCS).

* Although these macro instructions are available to all users, they are employed primarily by system programmers; therefore, refer to System Programmer's Guide, Form C28-2008, for a detailed description of these macro instructions.

TRANSFER TO COMMAND MODE FROM PROGRAM MODE

TSS/360 provides a user with several ways of interrupting a program's execution, either temporarily or permanently, and passing control to command mode for subsequent processing. The following macro instructions are available to users who want to transfer from program mode to command mode.

PAUSE (for conversational tasks only) types out a user specified message at the user's terminal, passes control to command mode enabling a user to enter commands at his terminal. After each command is issued, the system prompts the user for the next command. To resume program execution, the user must issue a RUN command at the terminal.

COMMAND similar to PAUSE macro instruction except it can be issued in nonconversational as well as conversational tasks. In conversational mode it has the same effect as PAUSE. In nonconversational mode, the specified message is written out on the tasks SYSOUT data set. The SYSIN data set is then polled for the next commands. The program's execution can be resumed by issuing a RUN command either at the terminal or in the SYSIN data set.

EXIT notifies the system a task has reached a logical conclusion (terminating point) by writing both a predefined system message and a user specified message on the SYSOUT device. If EXIT is issued in a program being executed in a conversational task, the messages are written on the users terminal, control is passed to command mode, and the next commands are taken from the terminal. In nonconversational mode, the messages are written on SYSOUT and the next commands are read from SYSIN.

ABEND serves as an error exit for an assembled program. There are three error exits available; 1) terminate execution of the program and return to the users terminal or to TSKABEND, 2) delete the users task from the system, initialize a new task, and pass control to command mode where commands can be entered for the new task, 3) delete the user's task from the system and deactivate his terminal.

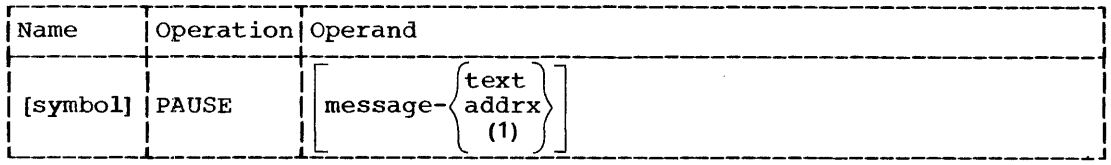
OBEY allows a user to temporarily pass control to command mode, execute a specified command or command statement, and continue processing at the next sequential instruction in his program.

Detailed explanations of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to transfers from program mode to command mode can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

PAUSE -- Enter Command Mode (R)

The PAUSE macro instruction switches a conversational task from program mode to command mode. A PAUSE macro instruction issued in a nonconversational task is ignored. During program stoppage, the user may issue whatever commands he wishes directly from the terminal. The task can be returned to program mode by issuing a RUN command.

The word PAUSE and the optional message specified in the operand are displayed on SYSOUT.



message
specifies the message to be issued. If an addrx is given, it must point to the location in storage which contains the message as a character string. If (1) is written, the location of the character string must be in register 1 before execution of the macro instruction. The first byte of the message must contain the length, in bytes, of the message.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: If the user has control of interruptions before issuing a PAUSE macro instruction, the system regains control of them until a RUN command is issued.

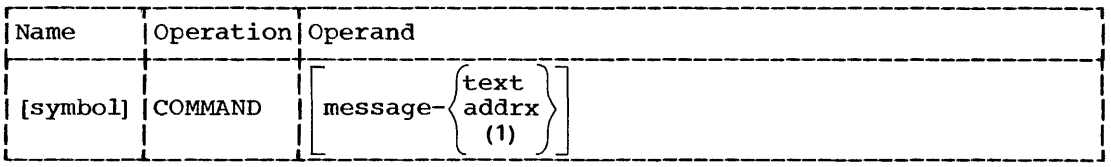
EXAMPLES: In EX1 the message is supplied as text. In EX2 the message is given at location DARRY.

```
EX1  PAUSE  'PROG DECISION AT STMT LOOP3'
EX2  PAUSE  DARRY
```

COMMAND -- Enter Command Mode (R)

The COMMAND macro instruction switches the task from program mode to command mode to allow the user to enter commands. This macro instruction causes an unconditional pause, and executes whether the task is conversational or nonconversational. Any commands may be issued from the terminal during conversational program stoppage. If the stopped program is nonconversational, the SYSIN data set will be interrogated for the next commands. The task can be switched back to program mode by issuing a RUN command.

The word COMMAND followed by the optionally specified message is written on SYSOUT.



message
specifies the message to be issued. If an addrx is given, it must point to the location in storage which contains the message as a character string. If (1) is written, the location of the character string must be in register 1 before execution of the macro instruction. The first byte of the message must contain the length of the message (in bytes).

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: If the user has control of interruptions before issuing a COMMAND macro instruction, the system regains control until a RUN command is issued.

EXAMPLES: In EX1 the message is supplied as text. In EX2 the message is given at location BEMEL.

```
EX1  COMMAND    'PROG IN SUBRTN SQROOT'
EX2  COMMAND    BEMEL
```

EXIT -- Normal Program End (R)

The EXIT macro instruction terminates program execution and switches the task to command mode. The words "EXIT, RELEASE ALL UNNEEDED DEVICES" followed by the specified message are written on SYSOUT. No messages are written on SYSOUT if the NOMSG operand is specified.

Name	Operation	Operand
[symbol]	EXIT	[message-{text / addrx / (1)}] [,NOMSG]

message
: specifies the optional message to be issued. If the text option is chosen, the actual message, enclosed in apostrophes, is written in the operand field. If an addrx is given, it must point to the location in storage which contains the message as a character string. If (1) is written, the location of the character string must be in register 1 before execution of the macro instruction. The first byte of the message must contain the length (in bytes) of the message.

NOMSG
: specifies that no messages are to be printed on SYSOUT when the exit is taken.

PROGRAMMING NOTES: If EXIT is issued in a conversational task, the message is written on the user's terminal and the next commands taken from the terminal. If issued by a nonconversational task, the message is written on the SYSOUT data set and the next command is taken from the SYSIN data set.

The EXIT macro instruction returns control to the user's terminal; to return control to a calling module, the RETURN macro instruction must be used.

If the user issues a USATT macro instruction to get control of interruptions and later issues an EXIT macro instruction, the system gains control of interrupts and will not relinquish control until a return is made to the user program and another USATT macro instruction is issued.

EXAMPLES: In EX1, the user supplies the message text as a character string. In EX2, the message text is given at location MSGTEXT. In EX3, no messages will be printed on SYSOUT

```
EX1       EXIT      'COMPLETED ARDUOUS'
EX2       EXIT      MSGTEXT
EX3       EXIT      ,NOMSG
```

ABEND -- Abnormal Task End (R)

The ABEND macro instruction serves as an error exit for an assembled program, either to terminate execution of the program or to eliminate the user's current task from the system, and return control to the user in command mode.

Name	Operation	Operand
[symbol]	ABEND	compcode-{value / (0)},message-{text / addrx / (1)}

compcode
specifies the exit type as 1, 2 or 3. Compcode 1 causes return either to the conversational user at his terminal or, in nonconversational mode, to a data set with ddname TSKABEND, to retrieve commands for execution; in either case, the task is returned to command mode. Compcode 2 wipes out user's task. If the task is conversational, a new task is created for the user and turned over to him as though he had just logged on. Compcode 3 is similar to compcode 2 in that it wipes out the user's task, but it does not create a new task or return control to the user. The user terminal will be deactivated.

If (0) is written, the value must be loaded into register 0 before executing this macro instruction.

message
specifies the message text to be issued to SYSOUT when the ABEND macro instruction is executed. If an addrx is given, it must point to a one-byte length field that precedes the message text field; text length in bytes is expressed in hexadecimal. The message may be up to 257 bytes in length. If the text option is chosen, only the text need be specified; the number of characters in the message is not specified.

If (1) is written, the address of the length byte must be placed in register 1 before executing the macro instruction.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: ABEND with compcode 1 returns the task to command mode and removes any previously invoked user control of attention interrupts.

Although the VPSW and the general registers are displayed on SYSOUT together with the message text provided in ABEND, identifying error exits by providing discrete messages for different exits via ABEND may facilitate checking.

The TSKABEND data set might contain a sequence of PCS commands that obtains a selective dump of the program before terminating the task with a LOGOFF command.

If an error occurs during the processing of a compcode 1 condition, the ABEND procedure is reinvoked, and the error is processed as a compcode 2 condition.

EXAMPLES: The user wishes to provide an error exit if his program encounters trouble on one path. He includes in that path the ABEND macro instruction:

```
ERROR   ABEND    1,'ABEND BECAUSE TROUBLE IN PATH N'
```

The user wants to provide discrete messages for different error conditions. For his first error condition, he provides the message:

```
ERR1    DC       AL1(L'TEXT1)
TEXT1   DC       C'ABEND FOR INCOMPLETE PATH'
```

In the coding path that discovers this first error condition, he includes:

```
        LA       1,ERR1
        B        EREX
EREX    ABEND    1,(1)    COMMON ERROR EXIT
```

OBEY -- Execute a Command or Command Statement (O)

This macro instruction may exist anywhere in the programmer's code and allows him to execute a command or command statement even though not in command mode. Upon execution of the OBEY macro instruction, the command or command statement specified via the macro instruction operands is executed; then control is returned to the user's program.

Name	Operation	Operand
[symbol]	OBEY	com-{addr / text / (1)}

com
: identifies the text of the command or command statement either by specifying it directly as a character string enclosed in apostrophes, or indirectly as the address of the command or command statement character string which the user has coded elsewhere in his program, or as register notation. If no operand or (1) is written, the address of the command character string must have been loaded into register 1 before execution of this macro instruction.

PROGRAMMING NOTES: If the user chooses the addr form of the operand, the address he specifies must point to the first byte of the command or command character string, and the byte which precedes the character string must contain a count of the bytes composing the character string. No special alignment is required.

L - AND E- FORM USE: The L- and E-forms of this macro instruction are allowed and have no special requirements. The E-form of the macro instruction may specify any parameters; however, the parameters specified in the E-form will overlay those specified in the L-form. The E-form may not specify more operands than are specified in the corresponding L- form.

For example:

```
SUE       OBEY   COMADDR,MF=L
                  .
                  .
          OBEY   OTHERCOM,MF=(E,SUE)
                  .
                  .
          DC     AL1(L'COMADDR)
COMADDR   DC     C'EXECUTE PROG2'
          DC     AL1(L'OTHERCOM)
OTHERCOM  DC     C'EXECUTE PROG3'
```

When the E-form of this macro instruction is executed, the program specified via the L-form (PROG2) will be replaced in the command string by the program (PROG3) indicated via the E-form of the macro instruction.

EXAMPLES:

```
          OBEY   'PROCDEF PAR1'      Obey a command
          OBEY   'EXECUTE MYPROG'
          OBEY   'BACK THISPRG'
          OBEY   COMADDR
          OBEY                       (Where no operand implies
           .                         that register 1 has been
           .                         previously loaded with the
           .                         address of COMADDR and the
           .                         number of bytes composing
          DC     AL1(L'COMADDR)      COMADDR is in the byte
COMADDR   DC     C'EXECUTE PROGA'    preceding COMADDR)
```

CLIC* -- Read Command From SYSIN (Conversational) (O)

The CLIC macro instruction allows a conversational user to enter commands from his terminal during program execution. If the user has control of interruptions before issuing a CLIC macro instruction, the system regains control of them until a RUN command is issued. If this macro instruction is issued in a program operating in nonconversational mode, it is ignored.

CLIP* -- Read Command From SYSIN (O)

The CLIP macro instruction reads a command from the SYSIN device for either conversational or nonconversational tasks. If the user has control of interruptions before issuing a CLIP macro instruction, the system regains control of them until a RUN command is issued.

RTRN* -- Create Privileged Linkage Queue Entry (O)

The RTRN macro instruction causes a privileged linkage queue entry to the Director to be created, which logically completes a run.

*Although these macro instructions are available to all users, they are employed primarily by system programmers; therefore, refer to System Programmer's Guide, Form C28-2008, for a description of these macro instructions.

COMMUNICATION BETWEEN USER PROGRAM AND SYSIN/SYSOUT

The TSS/360 communication facilities permit a user to pass data, messages, and commands, to and from a user's SYSIN and SYSOUT devices. The macro instructions which effect communication between a problem program and these devices are:

GATRD reads a record from a SYSIN device, translates it to internal code, and places it in a programmer-designated area of virtual storage.

GATWR translates a record that is currently stored in a programmer-defined area and writes it onto a SYSOUT device.

GTWAR translates a record currently stored in a programmer-defined area and writes it onto a SYSOUT device; then, it reads a record from the SYSIN device and places it in another programmer-defined area of virtual storage.

GTWSR (for conversational tasks only) translates a record currently stored in a programmer-defined area and writes it on a SYSOUT device; then, it reads a record from the terminal and places it in another programmer-defined area of virtual storage.

SYSIN reads a line of data from a SYSIN device, translates it to internal code, and places it in a programmer-designated area of virtual storage, or, if the input line contains a command, or command statement, it transfers the line to the Source List for subsequent Command Analyzer processing.

PRMPT Invokes the TSS/360 User Prompter Facility and writes messages from a user or system defined message file to SYSOUT and, if specified, reads back responses from SYSIN.

MSGWR provided for compatibility with previous TSS/360 programs. It has the same function as the PRMPT macro instruction. The PRMPT macro instruction should be used in lieu of the MSGWR macro in future programs.

MCAST temporarily replaces the Character Translation Table (in the user's session profile) with a user specified Character Translation Table, enables a user to change control function characters such as a continuation character with a new character representation. The character Translation Table controls the transfer of data between SYSIN, a user's program, and SYSOUT when communication macro instructions such as GATRD, GATWR, etc. are issued.

A detailed explanation of the above macro instructions and the format in which they may be specified are shown below. Further information pertaining to communication between user programs and SYSIN/SYSOUT devices and the related macro instructions can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

GATRD -- Get Record from SYSIN (S)

The GATRD macro instruction reads a record from the user's SYSIN and places it in a specified area.

Name	Operation	Operand
[symbol]	GATRD	msg-addr,length-addr[,SIC]

msg
specifies the address of the area into which the input record is to be placed. The user must define the length of this area elsewhere in the program.

length
specifies the address of a fullword containing the length of the expected input record; the maximum length of line depends upon the input source:

VAM data set	128 characters
1050	130 characters
2741	130 characters
Model 35 KSR	80 characters

On return, the actual record length is stored at the same address.

SIC
indicates whether characters within SYSIN representing control functions (specified as such in the Character Translation Table, CTT) are to be regarded as input characters by the GATRD macro instruction. If SIC is specified, all characters in the CTT (located in the user's session profile) are translated to internal code and transmitted to storage regardless of the functional code assigned to it in the CTT. Absence of this operand requests the standard mode in which only characters assigned the translation code (00) in the CTT are translated and transferred to storage, while characters assigned to other functional codes are not transferred to storage. Thus, in the standard mode, characters within a line of SYSIN that are assigned unique functional codes in the Character Translation Table, such as the backspace or cancel control functions, will not be read into storage as part of the SYSIN input line.

Records whose length is less than one line (from the terminal keyboard), or one card image (from the terminal card reader) are read by one GATRD macro instruction. Records longer than one line or card image are truncated if continuation of the record is not indicated. If continuation is indicated, succeeding GATRD macro-instructions can be used to read the remainder of the record.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: The first GATRD (or GTWAR) issued fetches the first record of the specified length from SYSIN. For each subsequent record to be read from SYSIN, GATRD must be used. GATRD cannot be used to recover a record that was read by an earlier GATRD.

If the SIC operand is specified, both input characters, as well as control characters, must be included in the length count specified via the length operand of the GATRD macro instruction.

In the standard mode, a character is transmitted to the message area only if it satisfies the following four conditions:

1. It is assigned the translation code (00) in the Character Translation Table (see Command System User's Guide, Form C28-2001 for additional information pertaining to the CTT).

2. It is not deleted by the action of any characters that are assigned the backspace or cancel functions.

3. It appears on a record to the left of all characters assigned the end-of-message function.

4. Space is available for it in the area specified by the msg operand.

A record read in from SYSIN is truncated if it is longer than the length specified and contains no continuation character.

On return from GATRD, register 15 contains two bytes of coded information (hexadecimal) in bits 16-31, as shown in Table 5.

Table 5. Return Codes from All GATE Macro Instructions

Bits 16-23 Code	Significance
0	Input record contains no continuation code; record is therefore complete.
1	Input record contains a continuation code. Issue a GATRD to get next portion of record.
2	Record was truncated because it exceeded maximum length specified by the user.
Bits 24-31 Code	
0	SYSIN is VSAM (nonconversational).
4	SYSIN is VISAM (nonconversational).
8	Attention interruption occurred; record, if any, is unpredictable.
10	SYSIN received from terminal keyboard.
20	SYSIN received from card reader at terminal.

On return, the actual record length (in bytes) is stored at the same address at which 'length' was specified.

EXAMPLE: A 120-character record (i.e., 120 characters assigned the translation function within the CTT) is to be fetched from SYSIN and placed in the area READIN:

```
EX1     GATRD     READIN,ILENGTH
```

In this example, the user has defined length elsewhere in the program:

```
ILENGTH   DC      F'120'
```

Note that the absence of the SIC parameter defaults to the standard mode in which both input and functional characters are to be translated by the standard Character Translation Table and transmitted to the message area.

GATWR -- Write Record on SYSOUT (S)

The GATWR macro instruction writes a message on the user's SYSOUT from an area in storage.

Name	Operation	Operand
[symbol]	GATWR	msg-addr,length-addr[,SIC]

msg
specifies the address of the area containing the message text. Any characters that can be represented in the terminal character set are accepted, including blanks, parentheses, and commas.

length
specifies the address of a fullword which contains the length of the message to be issued. If the message is longer than the maximum line length of SYSOUT, GATWR will write as many lines (or records) as are necessary up to a maximum message length of 260 bytes. The user may specify where to end each record by inserting preferred breakpoint characters (X'72') in the message.

SIC
indicates whether characters in the storage area representing control functions (specified as such in the Character Translation Table, CTT) are to be regarded as output characters by the GATWR macro instruction. If SIC is specified, all characters in the CTT (located in the user's session profile) are translated from internal code and transmitted to the user's SYSOUT regardless of the functional code assigned to it in the CTT. Absence of this operand requests the standard mode in which only characters assigned the translation code (00) in the CTT are translated and transferred to SYSOUT, while characters assigned to other functional codes are not transferred to SYSOUT. Thus, in the standard mode, characters in storage assigned unique functional codes in the Character Translation Table, such as the backspace or cancel control functions, will not be translated to the user's SYSOUT as part of the output line.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: The return codes from GATWR are shown in Table 5.

If the SIC operand is specified, characters assigned the translation code (00) in the Character Translation Table, as well as characters assigned control function codes, must be included in the message length count specified via the length operand of the GATWR macro instruction.

EXAMPLE: A 16-character record is to be written on SYSOUT:

```
EX1     GATWR     RECOUT,LENGTH
```

In this example the user has coded elsewhere in the program:

```
RECOUT  DC        C'COMPLETED ROUND1'
LENGTH  DC        F'16'
```

GTWAR -- Write Record on SYSOUT and Read Response from SYSIN (S)

The GTWAR macro instruction writes a message on the user's SYSOUT, then reads the next record from the user's SYSIN into the designated area of the user's virtual storage.

Name	Operation	Operand
[symbol]	GTWAR	msgout-addr,lengthout-addr, msgin-addr,lengthin-addr[,SIC]

msgout
: specifies the address of the area containing the message text. Any characters that can be represented in the terminal character set are accepted, including blanks, parentheses, and commas.

lengthout
: specifies the address of a fullword that contains the length of the message to be issued. If the message is longer than the maximum line length for SYSOUT, GTWAR will write as many lines (or records) as are necessary up to a maximum message length of 260 bytes. The user may specify the text for each line by inserting preferred breakpoint characters (X'72') in the message.

msgin
: specifies the address of the area into which the input record is to be placed.

lengthin
: specifies the address of a fullword containing the length of the expected input record. On return, the actual record length is stored in the address specified by lengthin.

SIC
: indicates whether characters representing control functions (specified as such in the Character Translation Table, CTT) are to be regarded as valid message characters by the GTWAR macro instruction. If SIC is specified, all characters in the CTT (located in the user's session profile) are transmitted to and from storage regardless of the functional code assigned to them in the CTT. Absence of this operand requests the standard mode in which only characters assigned the translation code (00) in the CTT are translated and transferred to or from storage, while characters assigned to other functional codes are not transferred. Thus, in the standard mode, characters assigned unique functional codes in the Character Translation Table, such as the backspace or cancel control functions, will not be read into storage as part of the user's SYSIN or written from storage on a user's SYSOUT device.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: If the SIC operand is specified, characters assigned the translation code (00) in the Character Translation Table, as well as characters assigned control function codes, must be included in the message length count specified via the lengthout and lengthin operands of the GTWAR macro instruction. If a continuation is indicated (the record extends over more than one print line), the user must provide a GATRD macro instruction to fetch the next portion of the record.

An input record is truncated only if it is longer than the length specified and contains no continuation character. Truncation begins with the rightmost character.

At conclusion of execution of the GTWAR macro instruction, register 15 contains two bytes of coded information (hexadecimal) in bits 16-31; see Table 5 for these codes. If a GTWAR macro instruction is issued in nonconversational mode, it does not write a record on SYSOUT; it does, however, read the next record from SYSIN.

EXAMPLE: A 16-byte message (i.e., 16 bytes assigned the translation code in the CTT) is written on SYSOUT and a 120-byte record is read from SYSIN into an area called ADLE.

```
EX1     GTWAR     VICTOR,LARRY,ADLE,DAZE
```

In this example, the user has coded elsewhere in the program:

```
VICTOR  DC        C'COMPLETED FIRSTR'
LARRY   DC        F'16'
ADLE    DC        CL120
DAZE    DC        F'120'
```

GTWSR -- Write Record on SYSOUT and Read Record from Terminal SYSIN (S)

The GTWSR macro instruction writes a message on SYSOUT and reads a record from the terminal keyboard (SYSIN) in conversational tasks only. Use of this macro instruction in a nonconversational task causes termination of the task; however, the message is written on SYSOUT.

Name	Operation	Operand
[symbol]	GTWSR	msgout-addr,lengthout-addr, msgin-addr,lengthin-addr[,SIC]

msgout
: specifies the address of the area containing the message text. Any characters that can be represented in the terminal character set are accepted, including blanks, parentheses, and commas.

lengthout
: specifies the address of a fullword containing the length of the message to be issued. If the message is longer than the maximum line length for SYSOUT, GTWAR will write as many lines (or records) as are necessary up to a maximum message length of 260 bytes. The user may specify where new lines should begin by inserting preferred breakpoint characters (X'72') in the message.

msgin
: specifies the address of the area into which the input record is to be placed.

lengthin
: specifies the address of a fullword containing the length of the expected input record. On return, the actual record length is stored in the address specified by lengthin.

SIC
: indicates whether characters representing control function (specified as such in the Character Translation Table, CTT) are to be regarded as valid message characters by the GTWSR macro instruc-

tion. If SIC is specified, all characters in the CTT (located in the user's session profile) are transmitted to and from storage regardless of the functional code assigned to them in the CTT. Absence of this operand requests the standard mode in which only characters assigned the translation code (00) in the CTT are translated and transferred to and from storage, while characters assigned to other functional codes are not transferred. Thus, in the standard mode, characters assigned unique functional codes in the Character Translation Table, such as the backspace or cancel control functions, will not be translated from storage to the user's terminal or and will not be read into storage as part of a terminal input line.

CAUTION: If this module is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: If the SIC operand is specified, characters assigned the translation code (00) in the Character Translation Table, as well as characters assigned control function codes, must be included in the message length count specified via the lengthout and lengthin operands of the GTWSR macro instruction. At completion of execution of the GTWSR macro instruction, register 15 contains two bytes of coded information (hexadecimal) in bits 16-31. These codes are shown in Table 5; however, since this macro instruction cannot be executed in nonconversational mode, the nonconversational mode return codes are not issued.

On return, the actual length of the input record (in bytes) is stored at lengthin.

If a continuation is indicated (the record extends more than one print line), the user must provide a GATRD macro instruction to fetch the next portion of the record.

An input record is truncated only if it is longer than the lengthin specified by the user and contains no continuation character. Truncation begins with the rightmost character.

EXAMPLE: In the following example, a 16-byte message is written on SYSOUT and a 120-byte record is read from the user's terminal (SYSIN) into area READIN:

```
EXI         GTWSR     OAREA,OLENGTH,READIN,ILENGTH
```

In the example, the user has coded elsewhere in the program:

```
OAREA       DC        C'COMPLETED FIRSTR'
OLENGTH     DC        F'16'
READIN      DC        CL120
ILENGTH     DC        F'120'
```

SYSIN -- Obtain a Message From SYSIN or the Source List (S)

The SYSIN macro instruction services the program in which it appears by providing information about the current operating task. This is done by retrieving input (i.e., either a command or data) from the Source List or the SYSIN for the task.

Name	Operation	Operand
[symbol]	SYSIN	msg-addr, length-addr [,source-code] [,prmpt-addr] [,exit-addr]

msg
specifies the origin of a user storage area in which the SYSIN macro instruction is to store the requested input message. No boundary requirements exist for this operand.

length
address of a full word which specifies the number of bytes in the user's storage area. If the requested input message is too long for the specified storage area, it is truncated on the right. The four low-order bits of register 15 contain a return code of 4 if truncation occurs. If the requested message length is less than the number of bytes in the user's storage area, the contents of this fullword are replaced by the actual number of bytes transmitted.

source
specifies a two byte code indicating the source from which the input message is to be obtained and the location to which it should be transmitted. The first byte serves as the source code and the second byte indicates the location code. If only the source code is specified, the second byte is left blank, and the transmittal location is as indicated under the various source codes. The source and location codes are described below.

Source code	Meaning
L	Obtain the input message line from the Source List (Created by the system Command Analyzer routine) and return normally if the line contains a message or data. If the message is a command, a return is made to the specified exit address without obtaining the command. If the message is a command but no exit address is specified, the program is abnormally terminated.
G	Obtain the message from SYSIN and, if it contains a message or data, return normally. If the message is a command, transfer the command to the Source List, but do not transmit it to the user specified storage area. If the message is a command but no exit address is specified, transfer the command to the Source List and terminate the program normally.
E	Obtain the input message from either SYSIN or the Source List, depending on the setting of the parameter, SYSINX, established in the user's profile by previous issuance of a DEFAULT command. The value of the SYSINX parameter in the user profile might have been previously established as either G, L or E. If the source operand is defaulted when issuing the SYSIN macro instruction, the source code existing in the user profile establishes the actual default source. SYSINX is initially set to G by the system.

Location Code

S	This code may be used as a suffix to any of the first three codes, but may not be used by itself. It modi-

fies the action of the code to which it is suffixed by causing commands to be transmitted to the user's storage area just as ordinary data or message input would be.

prmpt
: specifies a special command prompt string that is to be issued at the user's terminal to prompt the user to enter an input line. The indicated prompt string should be preceded by one byte containing the string length.

exit
: specifies the address that is to receive control if the requested message is a command and the source operand is not specified with the S as a suffix. This operand is not valid if the source operand is specified as LS, GS, or ES.

PROGRAMMING NOTES: The SYSIN macro instruction either prompts the user's SYSIN device for an input line or it reads a line from the Source List. When the input line is read, it is examined to determine if it contains commands, an input message, or data. If an input message or data was read from SYSIN or the Source List, the input line is transferred to the user specified input message area and execution of the user's program continues. However, a user may have entered a command or command statement in response to the prompting produced by execution of the SYSIN macro instruction. If the reply from SYSIN or the Source List is a command, or command statement, it is not transmitted to the user specified input area unless the suffix S appears in the code. Instead, SYSIN the routine passes control to the user indicated exit address. At his exit address, the user can then examine the commands by searching the Source List and either execute them immediately and continue processing, or execute them further on in his program.

A user can alter the action of the SYSIN routine by entering the command language prefix character (i.e., usually an underscore) following the SYSIN macro instruction routines prompt string when it is written out at the terminal. If commands are entered in this manner they are executed immediately and the SYSIN routine will return a code of 08 in register 15 to the user.

If the suffix S is used, commands will be transmitted to the user's input message area just as data normally is.

If a message is requested from the Source List when the latter is empty, the message is obtained from SYSIN instead.

The following return codes are placed in register 15 when control is returned to the user:

Code	Meaning
08	an immediate command (i.e., a command preceded by the system's command prompt string) was detected and executed.
10	the input line was in keyboard format; a normal return was made.
14	the input line was in keyboard format; the line was truncated.
20	input line was in card reader format; a normal return was made.
24	input line was in card reader format; the line was truncated.

When a normal return is made, the total number of bytes transmitted to the user area is passed to the user in the area in which he indicated the maximum message length.

L- AND E- FORM: Both the L- and E-forms of the SYSIN macro instruction are available and have no special requirements. The E-form of the macro instruction may specify any parameter; however, the parameters specified in the E-form will overlay those specified in the L-form. The E-form may not specify more operands than are specified in the corresponding L-form.

For example:

```
              .
              .
              .
  SUE     SYSIN    INAREA,LENGTH,G,PMPT,EXITEND,MF=L
          SYSIN    LENGTHB,,PMPTB,,MF=(E,SUE)
              .
              .
              .
```

When the E-form of this macro instruction is executed, the length and prompt string parameters (LENGTH,PMPT) specified in the L-form will be replaced in the parameter list by the length and prompt string (LENGTHB, PMPTB) specified in the E-form.

EXAMPLE: Execution of the following example will cause the prompt string 'ENTER I/D' to be displayed at the user's terminal, and his reply to be read from the terminal and transmitted to the user storage area labeled INAREA. The number of bytes transferred to INAREA would be placed in the LENGTH field specified by the user. When the SYSIN routine returns control to the user's program, register 15 will contain a hex return code of 10.

```
            .
            .
          MVC     LENGTH,LCON
          SYSIN   INAREA,LENGTH,G,PROMPT,EXITADR
            .
            .
EXITADR   RETURN
            .
            .
          DC      AL1(L'PROMPT)                  Length of prompt string
PROMPT    DC      C'ENTERID'
LCON      DC      AL1(L'INAREA)
LENGTH    DC      X'00'                          Length of INAREA
INAREA    DS      CL20
```

PRMPT -- Prompt System to Display a Particular Message (S)

The PRMPT macro instruction requests that the message associated with a particular message ID be displayed at the terminal and calls upon a system control program (User Prompter) to expedite the request.

Name	Operation	Operand
[symbol]	PRMPT	msgid-{addr / text}, resp opt-{N / P / U}, user resp-addr [, ([{param-{addr / text}} ..].)]

msgid
an original 8-byte message identification code associated with a message residing in a user provided message library. If the addr form is used it must point to an 8-byte field containing the msgid left adjusted and filled out with blanks.

resp
a one byte code indicating the types of responses, if any, the User Prompter program should expect to have furnished from the terminal when the message is displayed at the terminal. This code can be specified as:

N no response should be expected from the terminal. This option causes the User Prompter to display the message at the terminal and return control to the program containing the PRMPT macro instruction, after placing zero in the user response code field.

P a predefined response should be expected from the terminal. This option causes the User Prompter to display the message at the terminal, read a user response from the terminal, and then compare the user response to an expected response that was predefined in the message library. If a matching response is received, a code attached to the predefined response in the library is returned to the caller in the user response field defined by the PRMPT macro instruction. If a matching response is not found, the user is prompted with all of the predefined responses to terminal responses. For conversational tasks, if the next response is also improper, the user response field is set to zero and an error is indicated in register 15. Control is then returned to the user's program.

U An unpredictable response other than those defined in the message file, such as a string of information, should be expected from the terminal. This option causes the User Prompter to display a message at the terminal and then read an undefined response from the terminal. For example, the message might be "Enter User ID" to which the response would be an actual unique User ID value. In this case the User Prompter places a pointer to the response read from the terminal, in the user response field. The byte preceding the string must equal the length of that string (255 bytes maximum length).

user resp
A one word field in which the User Prompter indicates the type of user response to a message. For predictable responses, a unique predefined response code indicating which of the possible predefined responses has been entered, is placed, right justified, in the field by the User Prompter. For unpredictable responses a pointer to the response string is placed in the field.

param
information that is to be used to complete or alter the message being displayed at the terminal. Parameters can be specified as relocatable symbolic pointers or by register notation (addr), or as quoted strings (text). Parameters are separated by commas. If symbolic pointers are used, the strings to which they point must be defined in the caller's program as assembly language character constants and immediately preceded by a byte containing the length of the string. The number of parameters cannot exceed 20.

PROGRAMMING NOTES: The User Prompter is a centralized message storage display, explanation, and response handling facility available to both the system and user programmers. The message file, referenced by the User Prompter, must be set up via the standards defined in IBM System/360 Time Sharing System: Command System User's Guide, Form C28-2001, in order to allow for use of the User Prompter facility. All predefined responses set up in the message file should be preceded by a unique identification code.

Explanations of messages displayed at the terminal can be requested by use of the EXPLAIN command (see the Command System User's Guide for further details).

When control is returned from the User Prompter to the program containing the PRMPT macro instruction, register 15 will contain one of the following return codes:

Code	Meaning
0	No errors. Normal return
4	An I/O error has occurred
8	System error has occurred
12	Message could not be found
16	Message filtered out by user due to message display level choice specified by the user in the user profile or because of the length specified for the choice
20	Insufficient output buffer space available. Message truncated.
24	Explanation could not be found in the message file
28	Matching response not found among predefined responses in the message file
32	Invalid response option was specified. Response option not N, P, or U.
36	Message continued
40	Attention interrupt occurred during I/O operation

L- AND E-FORM USE: The L- and E-forms of this macro instruction are allowed and have no special requirements. The E-form of the macro instruction may specify any parameters; however, the parameters specified in the E-form will overlay those specified in the L-form. The E-form may not specify more operands than are specified in the corresponding L-form.

For example:

```
SUE          PRMPT     MSGIDADR,N,,MF=L
                .
                .
                .
             PRMPT     'MSGIDB',P,RESP,MF=(E,SUE)
                .
                .
                .
MSGIDADR     DC        C'MSGIDA'
                .
                .
                .
RESP         DS        F
```

When the E-form of this macro instruction is executed, the message (i.e., the message whose ID is MSGIDA) is replaced in the parameter table generated by the L-form macro expansion, by the message specified in the E-form (i.e., the message whose ID is MSGIDB). In addition a predictable response will be expected by the User Prompter instead of no response at all.

EXAMPLES: Both the user code, the message written at the terminal, and a user response, are displayed in the examples below:

```
EX1;    User Program:                   PRMPT 'CZATF',N,
        Message to Terminal;            CZATF ILLEGAL COMMAND

EX2;    User Program:                   PRMPT 'CZASE101',P,RESP
                                          .
                                          .
                                          .
                          RESP          DS     F
        Message to Terminal:            CZASE101 ENTER DSORG
        Response from Terminal:         VS

EX3     User Program:                   PRMPT 'CZABC201',U,IDCODE
                                          .
                                          .
                                          .
                                          .
                          IDCODE        DS     F
        Message to Terminal:            CZABC201 ENTER USER ID
        Response from Terminal:         EJB107

EX4     User Program:                   PRMPT 'CZSEB',N,('100',PARADD1)
                                          .
                                          .
                                        DC     AL1(5)
                                          .
                                          .
                          PARADD1       DC     C'CZATF'
        Message to Terminal:            CZSEB LINE 100 IN REGION
                                              CZATF DOES NOT EXIST
```

In EX1 no response from the terminal is expected and control is returned to the user's program after the message is displayed. EX2 expects predefined responses to be entered at the terminal. When the user responds by entering VS, the PRMPT routine searches a list of predefined responses (via the User Prompter) to find a match. A unique code, identifying which of the possible predefined responses the user has entered, is stored in the full word parameter (RESP) in the user's program and control is returned to the program. EX3 expects unpredictable responses to be entered at the terminal. When the user responds by entering his user ID, the routine determines if a valid ID has been entered, and if valid, places a pointer to the area in which the response is stored, in the user response option field (IDCODE). EX4 requires no user response, but, inserts variable message data ('100' and the string at PARADD1) parameters into the message contained in the message file and writes the completed message to the terminal. Thus, if the message is recorded in the messagefile as, LINE $1 IN REGION $2 DOES NOT EXIST, the variable entries $1 and $2 are replaced by the parameters provided via the operands of the PRMPT macro instruction as indicated in EX4.

MSGWR -- Issue Message and Get Response (S)

The MSGWR macro instruction issues a message to SYSOUT and, if specified, fetches the response and places it in a user-designated area. SYSOUT receives all the system messages such as diagnostics, responses, etc. In conversational mode SYSOUT is considered to be the terminal and in nonconversational mode, SYSOUT is considered to be the data set containing system messages that will be printed at the end of the task. Responses can only be made in conversational mode. Therefore, the response option may only be specified in conversational mode. If the

response option is specified in nonconversational mode, the task is abnormally terminated.

MSGWR (unlike the similar macro instructions, GATWR and GTWSR) issues system messages only. The text and the message numbers assigned to these messages are described in the document IBM System/360 Time Sharing System: Command System User's Guide. The user can modify a standard system message by inserting variable information into it. Variable fields are filled in by MSGWR, using information supplied by the user.

The format for the MSGWR macro instruction is indicated below. The information associated with each parameter must be placed in storage by the programmer before issuing the macro instruction.

Name	Operation	Operand
[symbol]	MSGWR	msgcode-addr,[varinf-addr,] [rarea-addr,rlength-addr]

msgcode
specifies the address of a full word containing the message number, a response flag, and the number of variable strings of text to be inserted. This information must be inserted in a full word by the user in the format described below.

Byte	Contents
0-1	message number - four hexadecimal digits
2	response flag - 1 if response is desired 0 if not
3	numbers of strings of variable text to be inserted

varinf
specifies the address of one or more double words which identify the text to be inserted in the variable field of the message. There are as many doubleword entries as are specified in byte 3 of msgcode. The first of these words given the number of bytes of text, in binary. The second word points to the actual text.

rarea
specifies the address of the area into which the response (if any) is to be placed.

rlength
is the address of a 1-word field into which the length of the response (if any) is to be placed. The response area must be large enough to accept the longest expected reply. The longest possible reply is 128 bytes. If the response does not fit in the allotted area (because less than 128 bytes were specified), it will be truncated, starting with the rightmost character.

If the user has any doubt about the length of the response, he should give 128 as the rarea. This is the maximum print line length and will thus prevent accidental truncation of an input record.

PROGRAMMING NOTES: On return from MSGWR, a hexadecimal code is loaded into the low-order byte of general register 15. The significance of these codes is as follows:

Code	Significance
0	No attention interrupt; no error in response length (if applicable).
4	Response too long for area specified. Truncation occurred.
8	Attention interrupt occurred; status of response (if any) is unpredictable.

Upon return from MSGWR, if a response was requested, the actual byte length of the response is placed in rlength.

General registers 2 through 12 and the floating-point registers are unaffected by expansion of the MSGWR macro instruction.

EXAMPLE: In the following example, the system message D001 is to be written on the terminal with a variable field containing XXUSERID; the expected response should contain a maximum of 8 characters, which is to be placed in an area called READIN.

```
EX1     MSGWR MSGCD,VARFLD,READIN,RLENGTH
```

In this example, the user has provided information required by the MSGWR macro instruction elsewhere in his program through use of DCs. The parameters of the MSGWR macro instruction have been specified using the symbolic addresses pointing to the DCs.

```
MSGCD      DC    X'D0010101'
VARFLD     DC    F'8'
           DC    A(TEXTVA)
TEXTVA     DC    C'XXUSERID'
READIN     DC    2F'0'
RLENGTH    DC    F'8'
```

MCAST -- Modify Character and Switch Table (O)

The MCAST macro instruction is used to temporarily replace the Character Translation Table (in a user's session profile) with a user specified Character Translation Table and temporarily overlay the control function characters such as continuation characters or end-of-block characters (also in the session profile) with new functional control characters. The CTT and the Profile Character and Switch Table in the session profile are both overlayed for the duration of the user's terminal session. If desired, the changes can be permanently recorded in the user's profile by issuance of the PROFILE command.

Name	Operation	Operand
[symbol]	MCAST	[CTT=addrx] [,EOB=addrx] [,CONT=addrx] [,CLP=addrx] [,TRP=addrx] [,DIV=addrx] [,SSM=addrx] [,USM=addrx] [,PL=addrx,CP=addrx] [,KC=addrx] [,RS=addrx]

CTT

address of a pointer to the 512 byte Character Translation Table which is to temporarily replace the one in the user's session profile.

EOB

address of the Source List end-of-block character that is to replace the one currently existing in the user's session profile. This character defines the end of an input block in the Source List to the Command Analyzer. The initial value is X'26'.

CONT

address of the continuation character that is to replace the one in the user's session profile. If the last character before a carriage return is a control language continuation character, the line of input is continued past the carriage return to include the next line entered at the terminal. The initial CONT character is defined as a hyphen, X'60'.

CLP

address of the control language prefix character that is to replace the one in the user's session profile. Entry of this character at the terminal by a user, requests the system to execute immediately the command following the character. Initially this character is defined as an underscore.

TRP

address of the transient command statement prefix character that is to replace the one in the user's session profile. When the user codes this as the first character of a command in a command statement, the SYSIN routine will recognize it as an immediate command and control will be passed to a predefind entry point in the language processor currently being executed. The language processor will then immediately process that command and either return control to the next sequential command in the command statement or perform other processing. The initial TRP character is a vertical line, X'4F'.

DIV

address of the preferred line divide character that is to replace the one in the user's session profile. A user can enter this character to indicate where he would prefer to have an input line broken if it becomes necessary or desirable to do so. The initial DIV character is defined as X'72'.

SSM

address of the new User Prompter system scope mask. This mask is used in conjunction with the explainable words of messages written to the terminal from the system message file. When the user requests an explanation for such a word (via the EXPLAIN command) this mask determines the pattern for searching through the hierarchy of word explanations in the message file. Each bit position in the one byte mask corresponds to a byte in an eight character label or message ID associated with a message containing the explainable word. Each bit that is set on (from right (7 bit) to left (0 bit)) causes a different level message file to be searched once. A complete scan is made and all indicated searches are executed.

The number of bytes in the message ID compared in each search is equal to the number of bytes to the left of the bit that is set on, plus 1, for the bit causing the search to be made. Thus if the 7 bit were set on, a search of 8 characters would be made; if the 1 bit were set on, a search of 2 characters would be made. The search for a particular level of explanation for a message begins by scanning the mask from right to left for on bits. If the first search doesn't locate the desired word, the scan continues to the next search indicating bit, etc., until the complete mask has been scanned and all levels of search have been completed. The initial default is defined as X'29'.

USM

address of the new User Scope Mask. Each bit represents a level at which a search and comparison is made to locate explainable words in a user defined message file (located in the user library). The user may set this mask according to his own search logic. See the SSM operand above for further information. The initial default value for USM is also X'29'.

PL

address of a byte containing the length of the Command Prompt String. This length cannot exceed 8. This length initially reflects the 3 character default value of the command prompt string operand. See CP below.

CP

address of a system Command Prompt String that is to replace the one in the user's session profile. This may be a string of up to eight characters. The initial default is an underscore followed by a backspace and a carriage return suppression character. The system uses this string to prompt the user to enter commands at the terminal. If this operand is specified, the PL operand must also be specified.

KC

address of a one byte keyboard/cardreader switch. This switch indicates the type of device from which input will be accepted by the system. It may be set with a K for a keyboard, or with an E to indicate either the keyboard or the card reader. E serves as the default parameter and causes the input device to be determined by examining the SYSIN parameter previously established in the user library by a DEFAULT command. The SYSIN parameter can be set to K or C; it is initially set to K. The KC operand is initially set to E.

RS

address of the carriage return suppression character that is to replace the one in the user's session profile. Normally a carriage return is executed after every message written on the terminal by the GATE routine, however, when this character appears as the last character in a message, no carriage return occurs. The next message written on the terminal begins where the last one left off. The initial value for RS is the colon, X'7A'. The suppression character is not written on SYSOUT.

CAUTION: If a user issues a PROFILE command via an OBEY macro instruction following MCAST, his user profile will be permanently changed. Users' should make certain, for subsequent program executions, that when communicating with those programs the updated control and functional characters are employed.

PROGRAMMING NOTES: The Character Translation Table consists of 512 contiguous bytes. The table is broken into two 256 byte sections. The first section contains the internal binary representation for each of the possible hexadecimal codes from 00 to FF, in sequential order. The second 256 byte section contains the function codes each displaced 256 bytes from its related hexadecimal translation code. The available function codes and the Character Translation Table are described in Appendix C of the IBM System/360 Time Sharing System: Command System User's Guide, Form C28-2001. A user must generate his new character Translation Table according to the prescribed format. A copy of the table can be found in that publication.

Since the MCAST macro instruction allows new interpretations for all current characters and control functions switches, it should be particu-

larly useful for publisher text editing applications where unique character interpretation is desired and line control changes are needed.

With varying line length capabilities of different devices, it may become necessary to divide a line of input. The DIV operand can be used to accomplish this. For ordinary printed text a user might make this character a space; then a line would be broken between words.

L- AND E- FORM USE: The L- and E-forms of this macro instruction are allowed and have no special requirements. The E-form of the macro instruction may specify any parameters; however, the parameters specified in the E-form may not specify more operands than are specified in the corresponding L-form.

For example:

```
SUE     MCAST   DIV=/,KC=K,MF=L
        MCAST   KC=E,MF=(E,)
```

When the E-form of this macro instruction is executed, the specification of the SYSIN device indicated via the L-form (K) will be replaced by the specification indicated in the E-form (E). Thus the system will accept input from the keyboard and card reader.

EXAMPLE: The user is replacing the Character Translation Table in the user's session profile with the characters indicated in the 512 byte table located at NEWTAB. In addition, the end-of-block character in the Profile Character and Switch Table in the user's session profile is being changed to an asterisk (*) and the command prompt string is being changed to a number sign (#).

```
            MCAST   CTT=NEWTAB,EOB=EOBCHAR,
                    PL=LNGTHCB,CP=NEWPRMPT
              .
              .
              .
              .
LENGTHCB    DC      AL1(L'NEWPRMPT)
NEWPRMPT    DC      C'#'
EOBCHAR     DC      C'*'
NEWTAB      DS      0CL512
              .
              .
```

COMMUNICATION WITH OPERATOR AND SYSTEM LOG

The TSS/360 communication facilities provide a user with macro instructions for communicating with the system log (a generation data group in which each VISAM data set contains a record of system to operator and operator to system communications for a startup to shutdown session), and the main operator's terminal. These routines should normally only be used for programs having specialized I/O routines that require operator intervention. The macro instructions providing this communication are indicated below.

WTO writes a user specified message on the main operator's console.

WTOR writes a user specified message on the main operator's console and reads a reply from the main operator console into a programmer-designated area.

WTL writes a user specified message on the main operator's console and records a copy of the user message in the system operator's log.

A detailed explanation of the above macro instructions and the format in which they may be specified are shown below. Further information pertaining to the communication between user programs, the main operator, and the system log, can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

WTO -- Write to Operator (S)

The WTO macro instruction writes a message on the main operator's console.

Name	Operation	Operand
[symbol]	WTO	message-text

message
specifies the message to be written on the operator's console. The message can include commas, blanks, and apostrophes as in a character constant.

The maximum message length is 256 bytes. The message does not include the required enclosing apostrophes.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: At completion of execution of the WTO macro instruction, the low-order byte of register 15 contains one of the following codes:

Code	Significance
0	Successful
4	Attention interruption
C	Invalid message length; no message sent.

L- AND E-FORM USE: The message operand is required in the L-form and is not permitted in the E-form.

EXAMPLE: In the following example, the message NOW COMPLETE is to be sent to the operator.

```
EX1     WTO       'NOW COMPLETE'
```

WTOR -- Write to Operator with Reply (S)

The WTOR macro instruction writes a message on the system operator console and enables the system operator's reply to be transmitted to the program issuing the macro instruction. No further processing of the program occurs until the operator replies.

Name	Operation	Operand
[symbol]	WTOR	message-text,reply-addr,length-value

message
: specifies the message to be written on the console. The message can include commas, blanks, and apostrophes as in a character constant.

  The maximum message length is determined at system generation time. This length must not exceed the physical line length on the console output device or 253 characters, whichever is less. The message appearing on the console does not include the enclosing apostrophes.

reply
: specifies the address of an area into which the message reply text should be placed.

length
: specifies the length, in bytes, of the reply text. The value must not exceed 256.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: At completion of execution of the WTOR macro instruction, the low order byte of register 15 contains one of the following codes:

Code (Hexadecimal)	Significance
0	Successful
4	Attention interruption
C	Invalid message length; no message sent.
10	Reply length greater than specified maximum reply length; reply was received, but only the maximum number of characters is in the reply area.

L- AND E-FORM USE: The message operand is required in the L-form and is not permitted in the E-form.

EXAMPLE: The message BEFF ON is written on the operator's console. The expected reply is four bytes long and will be stored at location ALPHA.

```
EX1     WTOR      'BEFF ON', ALPHA, 4
```

WTL -- Write to Log (S)

The WTL macro instruction writes a message in the system log and on the main operator's console.

Name	Operation	Operand
[symbol]	WTL	message-text

message
specifies the message to be written on the main operator's console and inserted in the system log. The maximum message length is 253 bytes. The message does not include the required enclosing apostrophes.

CAUTION: If this macro instruction is included in a module that is declared privileged (through use of the DCLASS macro instruction), the address of a save area must be placed in register 13 before execution of this macro instruction.

PROGRAMMING NOTES: The message can include commas, blanks, and apostrophes as in a character constant.

At completion of execution of the WTL macro instruction, the low-order byte of register 15 contains one of the following codes:

Code	Significance
0	Successful
4	Attention interruption
C	Invalid message length; no message sent.

L- AND E-FORM USE: The message operand is required in the L-form and is not permitted in the E-form.

EXAMPLE: The message ALL ON is to be sent to the operator and entered in the system log.

```
EX1     WTL       'ALL ON'
```

TIMING MAINTENANCE

There is a requirement within a time sharing system for the maintenance of various forms of elapsed time. The user requires the ability to set a timer which will measure the time of his task's execution or the elapsed calendar time. Three macro instructions have been provided by TSS/360 to provide the user with these abilities; the STIMER, TTIMER, and EBCDTIME macro instructions.

Each task has eight interval timers associated exclusively with that task. The STIMER and TTIMER macro instructions can set and test any or all of these eight timers. The EBCDTIME macro instruction can be used to record or measure the elapsed calendar time from a base or starting date of March 1, 1900.

STIMER sets a specified time interval into a timer, measures either task execution time or real clock time, and indicates what action should be taken when the time interval has elapsed.

TTIMER tests an interval timer previously set by the STIMER macro instruction and indicates the time remaining in that interval. It can also be used to cancel a previously specified timer setting.

EBCDTIME converts system maintained time into various EBCDIC formats, specified by the user. The time is expressed in some combination or years, months, days, ... and hundredths of seconds which have elapsed since the base date of March 1, 1900.

A detailed explanation of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to timing maintenance and related macro instructions can be found in IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032.

STIMER -- Set Interval Timer (O)

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.

Name	Operation	Operand
[symbol]	STIMER	{ TASK [,exit-symbol] , {DINTVL=addr | BINTVL=addr} { {REAL [,exit-symbol] | WAIT} , {DINTVL=addr | BINTVL=addr | TOD=addr | DO{W|M|Y}=addr} } [,TNO=0|integer]

TASK
specifies that the interval is to be decremented only when the task issuing the STIMER macro instruction is in control. A user may not request an interval timer to be set with a total task time greater than 7.5 hours.

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. This operand cannot be specified if WAIT is specified; if it is omitted when a TASK or REAL interval is specified, the task will be unaware of when the interval has ended.

DINTVL
specifies the address of a doubleword containing a decimal interval to be set into the timer. If real clock time was indicated (REAL), the doubleword must be aligned on a doubleword 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, where HH≤23, MM≤59, SS≤59, t≤9, and h≤9. If task time is being set (TASK), the maximum time interval that may be specified is 7.5 hours.

BINTVL
specifies the address of a fullword containing a binary interval to be established for this task. If real clock time was indicated (REAL), the fullword must be aligned on a fullword boundary and contain a positive 32-bit binary number in which the least significant bit has a value of 0.001 second. The specified interval must be less than 24 hours. If task time is being set (TASK), the maximum time interval that may be specified is 7.5 hours.

TOD
specifies the address of a doubleword containing the time of day at which the interval is to end. The doubleword must be aligned on a doubleword 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, an error message is issued at assembly time.

DOW
specifies the address of a four-byte field containing the day of the week at which the interval is to end. The four bytes must contain one of the following seven character combinations.

MOND
TUES
WEDN
THUR
FRID
SATU
SUND

DOM
specifies the address of a two-byte field containing the day of the month at which the interval is to end. The bytes contain two unpacked decimal digits which must take on a value in the range 01 through 31.

DOY
specifies the address of a fullword containing five packed decimal

digits of the form YYDDD, where YY= the last two digits of the year and DDD= the day of the year. These five digits are preceded by two packed decimal zeros and followed by a four-bit character such that all digits have the same zone if the 32-bit word is unpacked.

The TOD, DOW, DOM and DOY operands are meangingful only when a REAL or WAIT interval is specified. If a TASK interval is specified with one of these operands, an error message is issued at assembly time.

TNO

specifies the number of the programmed interval timer to be set. Nonprivileged programs may set timers 0 to 7. If this operand is omitted or invalid, timer 0 is assumed for nonprivileged programs.

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 doubleword 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 fullword 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 fullword 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 doubleword at LOC4.

```
EX1     STIMER    TASK,RTN1,DINTVL=LOC1
EX2     STIMER    REAL,RTN2,BINTVL=LOC2
EX3     STIMER    WAIT, BINTVL=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 interruption occurs, the exit routine (if specified) or the next instruction to be executed is given control when the task regains control normally.

When control is returned to the user program one of the following return codes is placed in register 15.

Code	Meaning
00	Normal return
04	Invalid time interval or invalid clock number was specified
08	Total user task time specified exceeds 7.5 hours. The timer was not set

Upon entry to the exit routine specified in an STIMER macro instruction, register contents are:

Register[1]	Contents
0	Pointer to save area
1	Parameter list pointer
2-12	Same as when the interruption occurred
13	Save area
14	Return address (internal task monitor location)
15	Address of the exit routine (this register can be used to provide addressability)

[1]See the STEC macro instruction for parameter list and pointer details.

The task monitor saves all registers internally. The exit routine can use all registers, except 14, without having to save and restore them.

Upon completion of the exit routine, the contents of register 14 must be as they were upon entry to the routine. The exit should terminate with a branch to the address in register 14.

TTIMER -- Test Interval Timer (O)

The TTIMER macro instruction provides the time remaining in the interval requested by a previous STIMER macro instruction and, optionally, cancels a previously specified timer interval.

Name	Operation	Operand
[symbol]	TTIMER	{TASK | REAL} [,CANCEL] [,TNO={0|integer}]

TASK
specifies a TASK interval, as specified in the associated STIMER macro instruction and as identified in the specified exit list.

REAL
specifies a REAL interval, as specified in the associated STIMER macro instruction and as identified in the specified exit list.

CANCEL
specifies that the identified interval should be cancelled. 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 executed, the CANCEL operand has no effect.

TNO
specifies the number of the programmed interval timer to be tested. Nonprivileged programs may test timers 0 to 15. Clocks 8-15 may be tested but they cannot be canceled; clock numbers over 15 are considered invalid. If this operand is omitted or invalid, timer 0 will be assumed for nonprivileged programs.

When control is returned to the user program, one of the following return codes is placed in register 15.

Code	Meaning
00	Normal return
04	Invalid clock number was specified

The time remaining in this interval is returned in register 0 whether or not the interval is canceled.

The remaining time appears as a 32-bit unsigned binary number in which the least significant bit has a value of 1 millisecond. The interval is returned in this form even if the interval was originally specified in decimal digits. If the interval expired and the event has already been dispatched before the TTIMER macro instruction was issued, a zero is returned in register 0.

EBCDTIME -- Convert System Time into EBCDIC Format (S)

The EBCDTIME macro instruction is provided to convert time from the format in which it is maintained by the system into various EBCDIC formats specified by the user. System time can be translated into any combination of years, months, days, hours, minutes, seconds, and tenths and hundredths of seconds by the EBCDTIME macro instruction.

Name	Operation	Operand
[symbol]	EBCDTIME	olist- {text / addr} ,time-addr ,L-integer

oplist
Can either be a string of special characters in text form or the address of such a string. The text constitutes a map containing special character groups which are converted into time and date; characters that are not part of a special character group are unchanged. The special character groups are are as follows:

CHARACTER GROUP	CONVERTED TO
YYY	year, from 1900 to 1999
YY	year, from 00 to 99
DDD	day of year, from 001 to 366
MO	numeric month, from 01 to 12
DD	Day of month, from 01 to 31
HH	Hours, from 00 to 23
MM	Minutes, from 00 to 59
SS	Seconds, from 00 to 59
SSS	Tenths of seconds, from 000 to 599
SSSS	Hundredths of seconds, from 0000 to 5999
MON	first 3 characters of month
DAY	first 3 characters of day
DAYW	first 4 characters of day

time

specifies the address of a doubleword binary number of microseconds to be converted to time and/or date. If the user provided the address of such a number, that number is converted into the format directed by the user specified map. If this parameter is defaulted, the system maintained time (i.e., a binary number of microseconds that have elapsed since March/1/1900) will be converted as directed by the user defined map. If the user supplied a time to be converted to a date, Mar 1, 1900, is used as the base for the conversion.

L

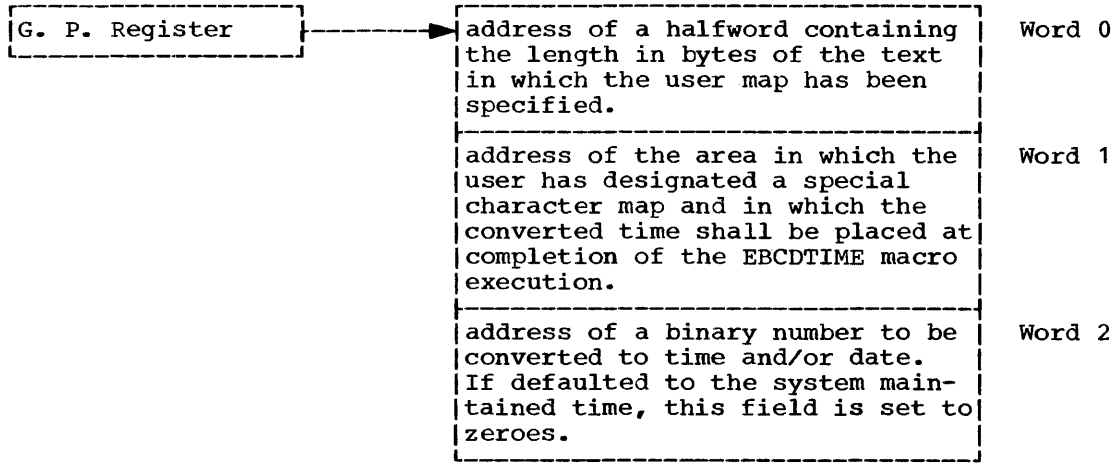
specifies a halfword containing the length of the map field (2 to 50 bytes). This parameter need only be specified if the addr form of the oplist parameter is used. If the text form of the oplist parameter is used, the length parameter should not be used. In such cases the system automatically calculates the length of the user defined map. When the length parameter is required, if it is specified as less than 2 or is not specified at all, a default map is provided which is 14 bytes long. The default map is: MO/DD/YY HH:MM. Normally, if the length parameter is greater than 50, the map is truncated on the right, however, if register notation is used for the length parameter, and L is greater than 50, the system will abend the task.

CAUTIONS: If register notation is used to indicate the length parameter, registers 1 and 14 cannot be specified.

Only upper case characters will be processed as part of a special character group.

If the oplist parameter is specified in the 'addr' format, then it should be reset after each use of EBCDTIME macro instruction because each execution of the macro would alter the map placed in the oplist.

PROGRAMMING CONSIDERATIONS: The parameter list generated by the EBCDTIME macro instruction is indicated below:



The length in bytes of the user map is placed in a halfword immediately following word 2 of the parameter list by the macro expansion. Similarly the user map is placed in a field immediately following the length field. If the user constructs his own parameter list, the bytes containing these parameters may be placed in other locations.

L- AND E - FORM USE: The L- and E- forms of the EBCDTIME macro instruction are allowed. They are written as described in 'S-Type Macro Instructions' in Section 1 except for the following exceptions. The oplist operand is required in the L-form of this macro instruction. In the E-form, the oplist operand is optional (i.e., if specified it usually points to an updated parameter list which is to overlay the parameter list indicated via the L-form of the macro instruction), but, if specified, only the addrx form of the operand can be used. The L-form macro instruction results in the generation of a parameter list in line. If the oplist parameter is not specified as text then the length of the text area should be specified. If the parameter oplist is not text and L is not specified then a default map of the following format will be assumed and, in addition, an error message indicating length was not specified will be generated:

```
MO/DD/yy HH:MM.
```

EXAMPLE:

```
EX1   EBCDTIME   PRINT1,L=22
```

In this example the user has defined PRINT1 elsewhere in his program as:

```
PRINT1  DC  CL22'THE  DATE  IS  DD  MON  YY"
```

On output, on the given date, PRINT1 would contain:

```
            THE DATE IS 24 FEB 67
```

```
EX2  EBCDTIME  'THE DATE IS DD MON YY'
```

REDTIM* -- Read Time (O)

The REDTIM macro instruction generates elapsed time (starting from March 1, 1900) as a double-precision fixed-point number representing year, month, day, hour, minute, second, millisecond, and microsecond.

*Although this macro instruction is available to all users, it is employed primarily by system programmers; therefore, refer to System Programmer's Guide, Form C28-2008, for a description of this macro instruction.

COMMAND CREATION

A user may desire to create unique commands that can be issued at his terminal. TSS/360 program management facilities provide several macro instructions to facilitate such operations. These macro instructions (BPKD and GDV) can be employed in a user coded routine that is to serve as the expansion of a command. The BPKD macro instruction must be used in conjunction with a BUILTIN command entered at the terminal. The functions of these macro instructions are summarized below.

BPKD written in a PSECT associated with a user coded routine that is to serve as the expansion of a command, being created by a user and generates all necessary linkage information and parameter storage areas required for use by a BUILTIN command used with the macro instruction. The label of the BPKD macro instruction must be specified as an operand of the BUILTIN command.

GDV locates the default values in a user library associated with a command. When an issued command has defaulted parameters, the command expansion routine associated with it can use the GDV macro instruction to locate the default values. The parameter name specified by GDV must be the same as a dummy parameter name indicated by a BPKD parameter.

Detailed explanations of each of the above macro instructions and the formats in which they may be specified are shown below. Further information pertaining to command creation and management and related macro instructions can be found in IBM System/360 Time Sharing System: Command System User's Guide, Form C28-2001.

BPKD -- Create a Builtin Procedure Key (O)

The BPKD macro instruction is used in conjunction with the BUILTIN command to identify a user coded routine as the routine that is to be executed to process a user created command issued at the terminal. The command which causes execution of the user routine defined by the BPKD macro instruction must be assigned a name by the BUILTIN command (See Command System User's Guide.)

Name	Operation	Operand
symbol	BPKD	ep-symbol[,({[param-relexp]},...)]

ep
specifies the symbolic name of the entry point to the user coded routine to be executed when a command, named by the BUILTIN command, is issued at the terminal.

param
specifies the address of a dummy parameter that defines an operand that is to be specified with the user created command. One 'param' operand must be specified in the BPKD macro instruction for each dummy parameter coded by the user. Each such dummy parameter defines both the position, and any related keyword, for a particular operand of the command.

Each user coded dummy parameter consists of a DC containing a keyword or a character string that can be related to the operand being defined. Each such DC must be preceded by the length of the character string in the DC (see Example below). Although each DC contains this keyword specification, the actual operand, when specified during command issuance at the terminal, need not be indicated by using this keyword; it can be specified positionally. The position of each 'param' operand specified within BPKD macro instruction determines the positions of these operands when positional notation is used.

PROGRAMMING NOTES: At assembly time the BPKD macro instruction generates a table containing all the necessary linkage information and parameter storage areas required for use by the BUILTIN command associated with the macro instruction. The generated table contains pointers to the object module that is to process the command. This module is referred to as the command processing routine or module. The generated table also contains adcons pointing to each of the dummy parameter defining constants coded by the user to reflect the number of parameters required by the command routine. Any keywords which a user may desire to associate with a particular parameter, when issuing the command at the terminal, should be placed in the parameter defining constant. The number of available parameters which may be specified in a command is determined by the number of dummy parameters defined in the BPKD macro instruction. Space is reserved in the generated table in which pointers to the parameters values specified by the user at the terminal can be recorded.

An ENTRY statement is generated in the table for the symbolic name of the BPKD macro instruction. The recorded entry address is then used to establish linkage between the user coded module and the BUILTIN command that defines that module as a command processing module and associates it with a particular command name. When the named command is issued, pointers to the actual values of any positional or keyword parameters specified at the terminal, are placed within the reserved storage areas within the generated table in the PSECT containing the BPKD macro instruction, register 1 is set to point to these pointers so that the data can be referenced by the command processing module. If any command operand defined by BPKD is omitted when the command is issued at the terminal, these pointers are set to point to any default values which may exist; if none exist, they are set to zeros. Control is then passed to the command processing module which is executed using the necessary parameter values to accomplish the desired goal. If no parameter names have been defined in the BPKD macro instruction, indicating that no parameters are to be specified when the command is issued, the expansion of the macro instruction does not set up dummy parameter areas or allow specifications of parameters within the command. Examples of the table generated by BPKD are shown under the Example paragraph below.

The BPKD macro instruction must be supplied in the object code as part of a PSECT associated with the command processing module and must have any expected parameters defined therein. The entire command processor could be assembled within the PSECT if so desired. If the command processor is assembled in a module different from the one containing the BPKD macro instruction, its entry point and PSECT symbols must be used as arguments of an EXTRN statement in the assembly containing the BPKD macro instruction.

When a user desires to provide parameters for the command he is creating, for each such parameter, the user must provide code in the PSECT indicating a character string (or keyword) to be associated with that parameter and the length of that character string. The parameter symbols of the BPKD macro instruction must also be the name fields of the user defined character strings (i.e., PAR1 DC C'KEYWORD, where PAR1 is a parameter symbol for BPKD). It is these parameter strings of key-

words that will be associated with the parameters of a calling command when the command is issued.

Any command procedures created by use of the BUILTIN command and the BPKD macro instruction are automatically recorded in the user's permanent profile when the BUILTIN command is executed and can be used in later terminal sessions.

EXAMPLE: If a user wants to execute the object program instructions in a particular CSECT when a command called TROT is issued at the terminal, he could indicate this by specifying:

In a PSECT associated with the command processor module:

```
          BPKLABEL  BPKD   EPPROC,(DPAR1,DPAR2)
          *         ENTRY  BPKLABEL
          *BPKLABEL DC     R(EPPROC)              ptr to Command
                                                    Processing Module
                                                    entry point
          *         DC     V(EPBPKMOD)            PTR to BPKD psect entry
                                                    point
Macro     *PARAM    DC     A(2)                   No. of possible
generated                                           parameters
table
          *         DC     2F'0'                  Initially, words of
                                                    binary zero for
                                                    each parameter
                                                    After command execu-
                                                    tion, ptrs to the
                                                    actual values
                                                    indicated at the ter-
                                                    minal
          *PARPTR   DC     A(DPAR1)               Dummy parameter
          *PARPTR   DC     A(DPAR2)                 defining parameters
                    .
                    .
                    .
                    DC     AL1(L'DPAR1)           coded by user for all
          DPAR1     DC     C'KEYWORD1'              desired command
                    DC     AL1(L'DPAR2)             operands
          DPAR2     DC     C'KEYWORD2'
```

At the terminal:

```
                    .
                    .
                BUILTIN   TROT,BPKLABEL
                TROT      KEYWORD1=70,KEYWORD2=5000
                    .
                    .       (OR IF POSITIONAL PARAMETERS
                    .       ARE SPECIFIED)
                    .
                TROT      70,5000
                    .
```

When TROT is issued, pointers to any keyword or positional parameters are placed in areas reserved within the BPKD macro expansion, making them availaable to the command expansion routine located at entry point CSECTMOD. The routine at CSECTMOD is entered, with register 1 pointing to BPKLABEL+12, which will contain pointers to the actual parameter values entered at the terminal (the length of these actual values can be found in the byte preceding the value location); then the routine is executed and control is returned to the user terminal.

GDV -- Get Default Value (R)

The GDV macro instruction searches the user library associated with the current user (i.e., a private library assigned to each user when he joins the system) to find any predefined parameter default values for system or user created commands.

Name	Operation	Operand
symbol	GDV	[com-{addrx / text / (1)}]

com
specifies the name of a particular command parameter as a character string enclosed in apostrophes, or specifies the address of the parameter. If the address of the parameter is specified, the actual parameter must be preceded by its length. If (1) is written, or if no operand is written, the address of the parameter must have been loaded into general register 1 prior to executing the macro instruction.

PROGRAMMING NOTES: The GDV macro instruction is useful within user coded command expansion routines for locating any default values in the user library associated with that command. When the command issued at the terminal includes defaulted parameters, the command processor routine might execute the GDV macro instruction to search for defaults previously defined in the user library.

For user created commands, the name of the parameter specified by the GDV operand must be the same as the parameter name indicated in a BPKD macro instruction.

If there is no predefined default value in the user library corresponding to the parameter name indicated in the GDV macro instruction, register 1 is set to zero and control is returned to the command expansion routine.

If the GDV routine finds a default value in the user library, the virtual storage address of the default value is placed in register 1 and control is returned to the command processing routine.

The actual parameter should be a user coded character constant containing any keyword that is to be associated with the parameter. The user may, however, indicate these parameters at the terminal either as keyword parameters or positional parameters.

L- AND E-FORM USE: The L- and E-forms of this macro instruction are allowed and have no special requirements. The E-form of the macro instruction may specify any parameters; however, the parameters specified in the E-form will overlay those specified in the L-form. The E-form may not specify more operands than are specified in the corresponding L-form.

For example:

```
SUE         GDV     'DPAR1',MF=L
            GDV     INSTEAD,MF=(E,SUE)
             .
             .
             .
INSTEAD     DC      C'DPAR2'
```

When the E-form of this macro instruction is executed, the parameter specified in the L-form (dpar1) will be replaced by the parameter specified in the E-form (dpar2).

EXAMPLE: If a user has created a command to be issued at the terminal, by use of the BPKD macro instruction and the BUILTIN command, the command processing routine coded by the user might employ the GDV macro instruction as described below.

Terminal Commands	User Coded Command Processing Routine
.	
.	CSECTA CSECT
	.
BUILTIN TROT,BPKLABEL	.
.	GDV DPAR2
.	.
DEFAULT KEYWORD2=200	PSECTA PSECT
.	.
.	BPKD CSECTA,DPAR1,DPAR2
.	.
.	.
.	.
TROT 50	DC AL1(L'DPAR1)
	DPAR1 DC C'KEYWORD1'
	DC AL1(L'DPAR2)
	DPAR2 DC C'KEYWORD2'

The command TROT is created by the user, and is issued with a defaulted second parameter. In such cases the command processing routine then executes the GDV macro instruction to search the user library for any defaulted value that may have been previously specified. When the default value is located the command processing routine can then insert the appropriate data into the parameter list generated by the BPKD macro instruction and continue processing. In the example above the defaulted DPAR2 operand is found equal to 200.

SYSTEM ORIENTED USER MACRO INSTRUCTIONS

In addition to those system oriented user macro instructions already indicated within the various functional groupings within the data set management, and program management sections of this publication, there are several other such macro instructions available. Although any non-privileged user can issue these macro instructions, they are meant for use, primarily, by the user's more sophisticated system programmers. For this reason, their availability is indicated below, but the detailed descriptions on their use have been placed in IBM System/360 Time Sharing System: System Programmer's Guide, Form C28-2032.

AWAIT* -- Tests for Event Completion and Return Control (O)

tests for completion of an input/output event and returns control to the task if the event is completed, or places the task in a delay state from which it will be removed when any task interruption occurs.

TWAIT* -- Tests for Completion of Event (O)

tests for completion of a given event and either returns control to the task, if the event is complete, or places the task in a delay state from which it is removed when any task interruption occurs. It also causes all pages for this task, which are on paging drums, to be moved to auxiliary disk storage.

VSEND* -- Inter-Task Communication (O)

sends a message from one task to another. The designated task is located and the message is queued on its task status index (TSI) as a task external interrupt.

VSENDR* -- Inter-Task Communication with Response (O)

sends a message to another task and waits for a reply from the receiving task.

XTRSYS* -- E xtract From System Table (O)

used to extract specific information from the system table.

XTRCT* -- Extract TSI Field (R)

extracts a TSI field specified by the user, for subsequent user examination.

XTRXTS* -- Extract From TSI (O)

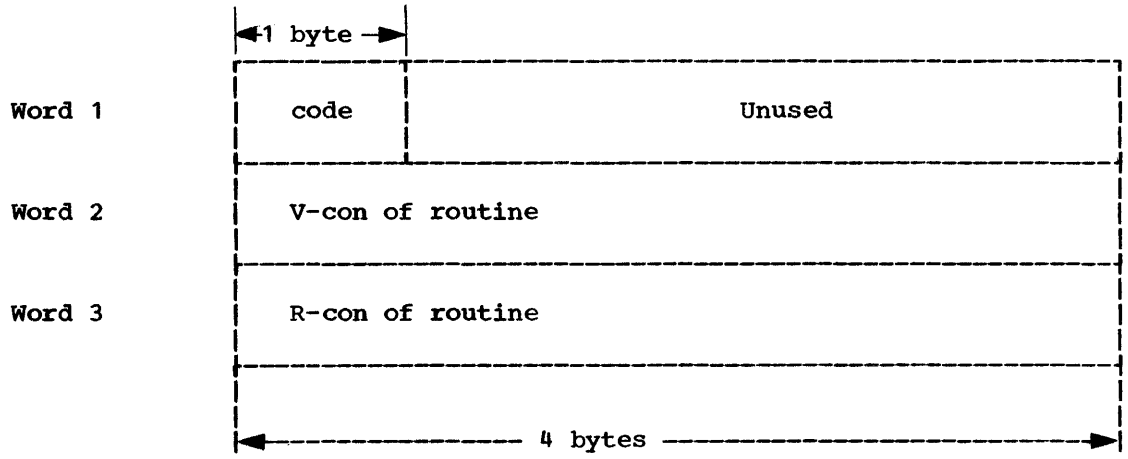
provides a means of extracting specific information from the extended task status index (XTSI).

*Meant for system programmers

The EXLST= operand of the DCB macro instruction may be specified only when using BSAM or QSAM data organization.

The exit list consists of a series of codes and addresses that inform the system of the location of a user-supplied exit.

Each entry in the list consists of three contiguous words aligned on fullword boundaries.



Entries do not have to be in order by code. To indicate the last entry in the exit list, the high-order bit of the byte containing the code is turned on.

If an exit routine is in the same assembly module as the exit list, two A-type address constants should be coded for words 2 and 3 of the entry.

Code (hex)	Meaning
00	Entry ignored, i.e., not active
01	User routine to process user header labels
02	User routine to create user header labels
03	User routine to process user trailer labels
04	User routine to create user trailer labels
05	User DCB exit routine
8X	Signal last entry in list, where X is 0-5 representing 00-05 codes above

Table 6. Conditions Upon Exit -- Routine Entries

	TRAILER LABELS[2]		HEADER LABELS[2]	
	Exit 03 INPUT	Exit 04 OUTPUT	Exit 01 INPUT	Exit 02 OUTPUT
OPEN				
INPUT			X	
OUTPUT				X
UPDAT			X	
INOUT			X	
OUTIN				X
RDBACK			X	
end-of-volume				
INPUT	X		X[1]	
OUTPUT		X		X[1]
UPDAT	X		X[1]	
INOUT (READ)	X		X[1]	
(WRITE)		X		X[1]
OUTIN (READ)	X		X[1]	
(WRITE)		X		X[1]
RDBACK	X		X[1]	
FEOV - CLOSE				
INPUT				X[1]
OUTPUT		X		X[1]
UPDAT				X[1]
INOUT (READ)				
(WRITE		X		X[1]
OUTIN (READ)				
(WRITE)		X		X[1]
RDBACK				X[1]

[1]Exit not taken if: (1) current volume in process is last volume, or (2) data set is to be closed

[2]If last I/O operation was backward, user header label routine is invoked for trailer label processing and user trailer label routine is invoked for header label processing

CHARACTERISTICS OF EXIT ROUTINES

Code 01: User Routine to Process User Header Labels

When this routine is entered, register 1 contains the address of the data control block being processed, and register 0 contains the address of an 80-byte buffer which contains a user header label. The first

If a data set to be written on tape was created via the DATA command, the first byte of each record contains an indicator for the origin of the record. Unless the startno operand is specified, this byte is written as part of the record upon issuance of the WT macro instruction. In such a case, if the record was originally entered through a card reader, the indicator byte will be written as a C. If it was entered through a terminal, the byte will be written as a blank character. When the startno operand is specified as 2 or greater, the indicator byte is bypassed and is not included as part of the written record. four bytes of the buffer contain the characters UHL1 to UHL8 or UTL1 to UTL8 depending on which of the eight permitted user header labels or trailer labels is being processed. To obtain the next label,

the user issues a RETURN (no operands except RC= are allowed) with a hexadecimal 04 in the low-order byte of register 15. When the last label is processed, the user issues a RETURN macro instruction with a hexadecimal 00 in the low-order byte of register 15. The user may not issue data management macro instructions for this data set in this routine.

Code 02: User Routine to Create User Header Labels

When this routine is entered, register 1 contains the address of the data control block being processed; register 0 contains the address of an 80-byte buffer in which the user is to build a label. The first four bytes already contain the characters UHL1 to UHL8 depending on which of the eight permitted user header labels is being created. These four bytes must not be altered. The user places information in bytes 4-79. Issuing a RETURN macro instruction, with a hexadecimal 04 in the low-order byte of register 15, causes the label to be written, and requests control to be returned to this routine so another label may be created. Issuing a RETURN macro instruction, with a hexadecimal 00 in the low-order byte of register 15, causes the last label to be written, and control is not returned to this routine. The user may not issue data management macro instructions to this data set in this routine.

Code 03: User Routine to Process User Trailer Labels

Same characteristics as Code 01.

Code 04: User Routine to Create User Trailer Labels

Same characteristics as Code 02, except the characters UTL1 through UTL8 are substituted for UHL1 through UHL8.

Code 05: User DCB Exit Routine

When this routine is entered, register 1 contains the address of the data control block being opened. The user may alter fields in the data control block, if desired. To return control to OPEN, the user issues a RETURN macro instruction (no operands except RC= are permitted) with a hexadecimal code of 00 in the low-order byte of register 15.

EXIT-LIST EXAMPLE

The following is an example of the coding of an exit list. The exit list must be in the same assembly module as the data control block (DCB macro instruction) which refers to it.

```
         DC       0F              ALIGN TO FULLWORD BOUNDARY
APPLE    DC       X'02'
         ADCON    IMPLICIT,EP=MHDRLAB
         DC       X'03'
         ADCON    IMPLICIT,EP=PHDRLB
         DC       X'01'
         ADCON    IMPLICIT,EP=PLABY
         DC       X'85'
         ADCON    IMPLICIT,EP=ALTER
```

The symbolic name of this exit list is APPLE. To use it, EXLST= APPLE must be written in the DCB macro instruction. Note that the high-order bit of the hexadecimal code for the last entry is on, indicating the last entry in the exit list.

APPENDIX B: SYNCHRONOUS ERROR EXIT ROUTINE (SYNAD)

When using BSAM, QSAM, VISAM (either for VISAM data sets or VISAM members of VPAM data sets), or IOREQ macro instructions, it is possible that errors may result from an attempt to process data; in many cases, certain remedial actions are available to the user.

If desired, a routine may be written for the purpose of receiving control from the system when an error occurs. The conditions, which cause control to be given to the SYNAD routine are described under each appropriate macro description.

The user indicates to the system that a SYNAD routine is supplied by writing the keyword parameter SYNAD= in the DCB macro instruction. The task is terminated if an error occurs which would normally cause SYNAD to be entered and no SYNAD was supplied.

The following is a list of suggested actions which may be taken in a SYNAD routine:

1. Issue a RETURN macro instruction, which causes a record to be accepted with error ignored (BSAM and QSAM only).

2. Set flags which are meaningful to the program.

3. Close the data set.

4. Resume processing at another point in the data set.

5. Call another routine.

6. Terminate the program.

Entry To SYNAD During BSAM or QSAM Operations

If BSAM or QSAM is being used, the contents of the general registers upon entry to the SYNAD routine are as follows:

Register	Bit	Usage
0	0 thru 31	Address of data event control block (DECB).
1	0	Set to 1, if error was caused by a READ macro instruction (for BSAM), or by GET or RELSE macro instructions (for QSAM)
	1	Set to 1, if error was caused by a WRITE macro instruction (for BSAM), or by PUT, PUTX or TRUNC macro instructions (for QSAM)
	2	Set to 1, if error was caused by a BSP, CNTRL or POINT macro instruction (for BSAM), or by a SETL macro instruction (for QSAM)
	3	Set to 1, if (1) error indicated by bit 0 did not prevent reading the block; or (2) if error indicated by bit 1 occurred during creation of a new block
	4	Set to 1, if request was illogical; e.g., a POINT macro instruction (for BSAM) or a SETL macro instruction (for QSAM) referred to a block not contained in the data set
	5 thru 31	Not used
2 thru 12		(contents that existed before the macro instruction was executed)
13		SYNAD R-con value (for BSAM), or address of service routines save area (for QSAM)*.
14		Return address
15		The address of the entered SYNAD routine

*For QSAM the word of the save area pointed to by GR13 will contain the PSECT address (R-con) for the SYNAD routine.

The DECB begins on a fullword boundary; its format is shown in Table 7.

Table 7. Data Event Control Block (DECB)

Byte	Bit	Usage
0	0	Always set to 0
	1	Completion flag; set to 1 when an I/O event is completed
	2 thru 31	(Used by the system)
4 and 5		Type field
6 and 7		Length field
8 thru 11		Data control block address
12 thru 15		Area addresss
16 thru 19		Pointer to status indicators
20 thru 25		(Used by the system)
26		Sense byte 1
27		Sense byte 2
28 and 29		(Used by the system)
30	1	Permanent error flag
31		(Used by the system)
32 thru 39		Channel status word

Type Field
contains a numeric value representing SF (for READ or WRITE) or SB (for READ).

Length Field
contains a binary number which represents the number of bytes in a block, or an indicator that the maximum block size specified in the data control block was used.

Data Control Block Address
contains the address of the data control block.

Area Address
specifies the I/O area address. For BSAM it contains the address of the high-order byte of an area in virtual storage which is the object of a forward READ or WRITE; or the address of the low-order byte of an area in storage which is the object of a backward READ operation.

Pointer to Status Indicators
contains the address of the status indicators, which are two bytes in the channel status word. If control is passed to the SYNAD routine status information (i.e., sense byte 1, sense byte 2, and the channel status word) is arranged as indicated above. Each of these status indicators is described in detail below.

Channel Status Word
is a doubleword illustrated below:

STORAGE PROTECTION KEY	COMMAND ADDRESS	STATUS BYTE 1 (unit)	STATUS BYTE 2 (channel)	COUNT-FIELD

The first six bits of sense byte 1 and all bits of status bytes 1 and 2 are device-independent. Their meaning is as follows:

Bit	Sense Byte 1
0	Command reject
1	Intervention required
2	Bus out check
3	Equipment check
4	Data check
5	Over run

Bit	Status Byte 1
0	Attention
1	Status modifier
2	Control unit end
3	Busy
4	Channel end
5	Device end
6	Unit check
7	Unit exception

Bit	Status byte 2
0	Program-controlled interruption
1	Incorrect length
2	Program check
3	Protection check
4	Channel data check
5	Channel control check
6	Interface control check
7	Chaining check

Bits 6 and 7 of sense byte 1 and all bits of sense byte 2 are device dependent. Refer to individual publications on specific devices for interpretation of these bits.

CAUTION: If any of the bits 4-7 of status byte 2 are on, the system cannot recover. Any subsequent I/O operations on the data set result in abnormal termination of the task.

If the permanent error flag in the data event control block is on, the program must not issue any further I/O operations to the data set.

If SYNAD is invoked because of SETL only the DCB address and status information in the DECB may be valid. All other fields may contain undefined information.

Entry to SYNAD During VISAM Operations

The SYNAD routine may be entered during VISAM operations (either processing of a VISAM data set or a VISAM member of a VPAM data set). The contents of the general registers upon entry to the SYNAD routine are as follows.

Register	Usage
0	Address of DECB, if error was caused by a READ or WRITE macro instruction. See Table 6
1	Address of the data control block
2 thru 13	(Contents that existed before the macro instruction was executed)
14	Return address
15	Address of the entered SYNAD routine

Additional information, concerning the error, that may prove useful to a SYNAD routine is found in the data control block fields, DCBEX1 and DCBEX2 (Appendix F).

When using data management services for input data sets, the exact number of records in the input data set need not be known. When the last record of a data set being sequentially processed is accessed, a subsequent attempt to access a record causes the system to transfer control to a specified point in the user's program. For BSAM the transfer is made when the READ macro instruction, which requested a block after the last block is accessed, is checked. For QSAM the transfer is made when the user issues a GET macro instruction after all the records in the data set have been processed.

The user indicates to the system where control is desired upon the end-of-data condition by writing the keyword parameter EODAD= in the DCB macro instruction. The task is terminated if an EODAD routine is not supplied and an attempt is made to access a record after the last record in the data set. For BSAM, the termination occurs upon issuing the CHECK macro instruction for a READ macro instruction issued after the last block of a data set is accessed.

When the end-of-data routine is entered, the general registers are set as follows:

Register	Usage
0	(Not defined)
1	Address of data control block
2 thru 12	(Same as before the routine was entered)
13	EODAD R-con value (for BSAM), or address of service routines save area (for QSAM) *
15	Address of EODAD routine

*The nineteenth word of the save area pointed to by GR13 will contain the PSECT address (RCON) for the EODAD routine.

APPENDIX D: CONTROL CHARACTERS

All record formats may optionally include a control character in each logical record. This control character is recognized and processed 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 alternatives are available; i.e., the control character may be in machine code or extended USASI code. If either option is specified in the data control block, the character must appear in every record.

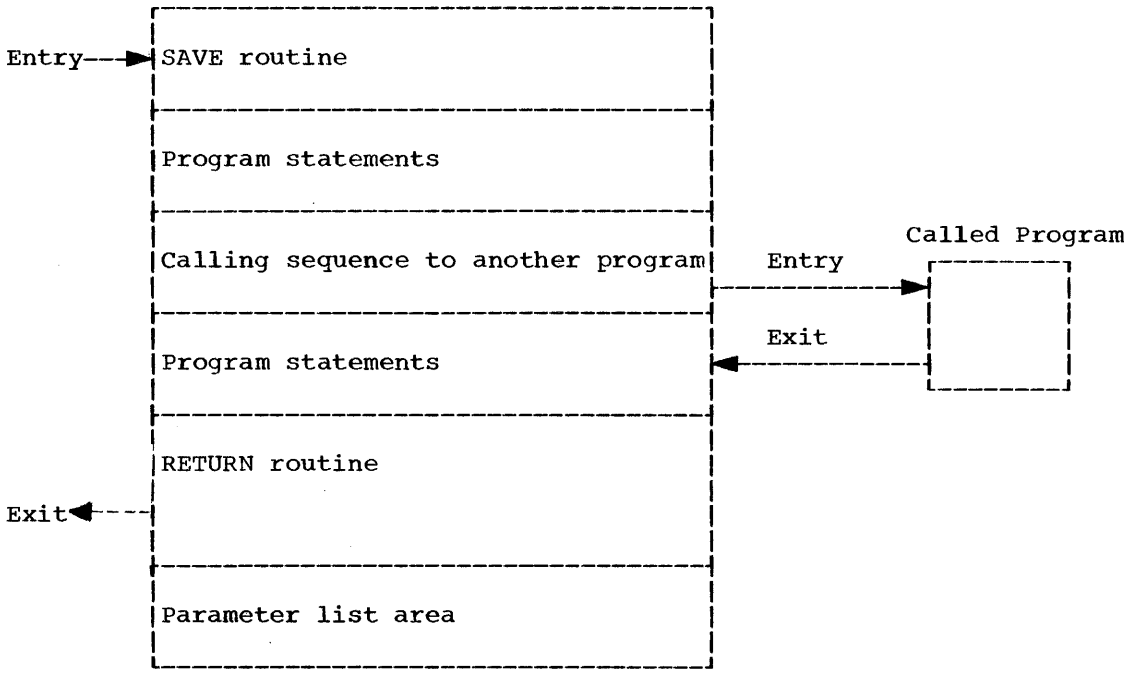
MACHINE CODE

The user may specify in the data control block that the machine code control character is placed in each logical record. The user-supplied byte must contain the bit configuration specifying a write and the desired carriage or stacker-select operation. Only those commands which include a write are permitted; the independent carriage and stacker select operations are excluded. Appendix F of IBM System/360 Time Sharing System: Command System User's Guide lists the machine codes.

EXTENDED USASI CODE

The user may choose to specify extended USASI code rather than machine code. Extended USASI code is the same as USASCII. The location of the byte is the same. This code byte must appear in each logical record if this option is chosen. The extended USASI codes are given in Appendix E of Command System User's Guide.

Linkage conventions govern communication among programs by establishing a standard which permits easy, efficient, error-free branching and linking to a desired program. The following chart summarizes the elements required by an assembler program to be both a calling and a called program:



In TSS/360, all linkage among programs residing in virtual storage conforms to one of the following three convention types:

- Type I -- Between two nonprivileged or between two privileged programs.
- Type II -- From a nonprivileged to a privileged program.
- Type III -- From a privileged to a nonprivileged program.

Only the Type I convention is presented in this appendix; Types II amd III conventions are described in System Programmer's Guide.

Type I linkage conventions include three basic standards to which the assembler user must adhere:

1. Utilizing the proper registers in establishing a linkage.
2. Reserving a save area in the calling program in which the called program may save the contents of the calling program registers.
3. Reserving a parameter area in the calling program, to which the called program may refer.

Proper Register Use

TSS/360 has assigned roles to certain registers used in generating a linkage. The function of each linkage register is illustrated below. Note that registers 2 through 12 are not used.

General Register	Usage
15,0	Supervisor Parameter Registers
1	Parameter List Register, Supervisor Parameter Register, or Parameter List Register
13	Save Area Register
14	Return Register
15	Entry Point Register, Return Code Register

It is the responsibility of the called program to maintain the integrity of general registers 2-12 so that their contents are the same at exit as they were at entry to the called program. It is the calling program's responsibility to maintain the floating point registers around a call. General registers 0, 1, and 13-15 must conform to the indicated conventions; when using system services (e.g., interrupt handling), these registers should not be used by the calling program, because their contents may be destroyed.

Reserving a Save Area

Every calling program must reserve an area of storage (save area) in which certain registers (i.e., those used in the called program and those used in the linkage to the called program) are saved by the called program.

The minimum amount of storage needed for the save area of a program that is both calling and called, is 19 words. Figure 2 shows the layout of the save area and the contents of each word.

A called program that does not call another program need not establish a save area. However, if registers 13 or 14 are used by the called program, that called program should save their contents in a desired location and restore them before returning control to the calling program.

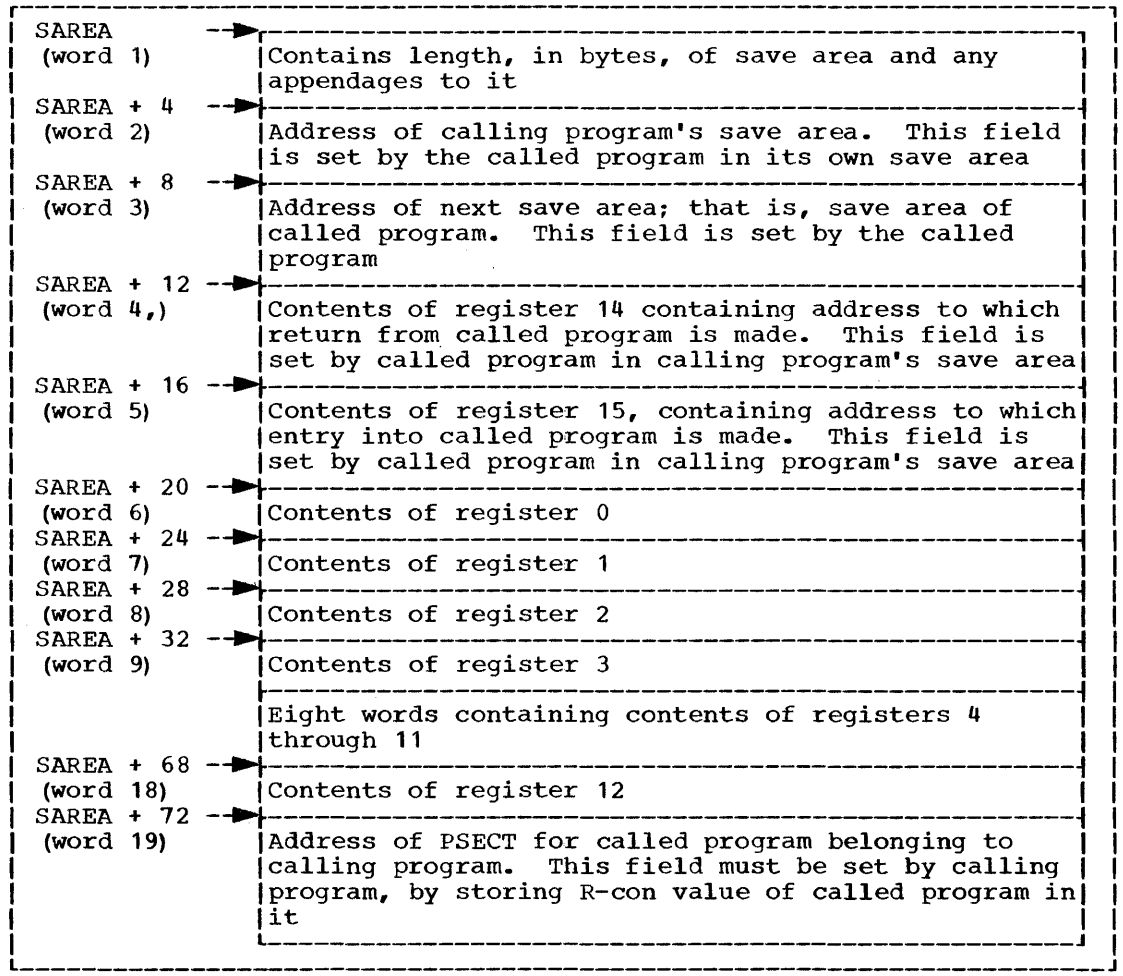


Figure 2. Save Area Layout and Word Contents

Reserving a Parameter Area

If a called program requires a parameter list, every program calling it must reserve an area of storage (parameter area) in which the parameter list used by the called program is located. Each entry in the parameter area occupies four bytes at a fullword boundary. If the parameter list is of variable length, the word preceeding the first entry contains the length (in words) of the parameter list. Each entry contains the address of an argument to be passed to the called program. The CALL macro instruction may be used to generate the parameter list as well as to link to the called program.

There are two types of linkage available to users of TSS/360: implicit linkage and explicit linkage. When an explicitly loaded module is no longer needed, it can be deleted explicitly.

Implicit Linkage

Program reference to a V- or R-type address constant of an external symbol constitutes a request for implicit linkage. When an undefined

external symbol is referred to in this manner, the loader is called to make available, in the user's virtual storage, those modules required tosatisfy this external reference. This automatic action requires only specification of external symbols and adcon types as required by the assembler.

Explicit Linkage

Within a given program there may be several references to different subprograms; however, for a given execution of that program, only one of those subprograms might be required. Since dependence on normal implicit linkage would require, in the calling program, the presence of adcons for all such subprograms, some unnecessary overhead would be experienced in preparing the unused adcons for linking.

It is also possible to develop, during program execution, the external name of the module, entry point, or CSECT which is to be explicitly linked. In this case, it may not be possible to specify the modules to be linked at assembly time.

To allow for these situations, two explicit functions are provided that retrieve the desired subprogram at object time. The LOAD macro instruction loads the desired program; the explicit CALL macro instruction, in addition to loading the program, establishes the necessary linkage to it.

Explicit Deletion

The DELETE macro instruction dispenses with a previously explicitly loaded program that is no longer needed, thereby making virtual storage available. In addition, any other program is deleted that is no longer required as a result of the deletion of a specified program.

This appendix contains descriptions of the contents of the fields of a data control block and the priority of the various sources for filling those fields, for those who desire to alter data control blocks or interrogate fields for the information contained therein.

Sources for Providing Data Set Attributes

In general, a user writes a source program to create or process data. This data is considered to be a data set. In TSS/360, the system requires that certain attributes and identifification information pertaining to data sets must be available to the system before a user can make use of the special programs and data management facilities comprising the TSS/360.

These attributes can be furnished to the system from two to six different sources depending on whether the data set being processed is a new data set or a data set that has been previously defined to the system. The combined information provided by these sources must provide the system with all the information it requires to begin processing a particular data set. The six possible sources which provide the system with the attributes of a data set are listed below in the order of their priority. Table 8 indicates the DCB operands applicable to each access method and their valid alternate sources prior to opening that data control block.

Source 1 - The User's Program

The user may alter or fill data control block fields any time after the block has been created by a DCB macro instruction. A DCB macro instruction with no operands merely reserves virtual storage for a data control block, with all its fields containing binary zero. The user has the opportunity to alter fields at OPEN time by specifying the address of a user routine which is to alter the DCB at open time as the EXLST parameter of the DCB macro instruction or lower priority DDEF macro instruction or command. Any user coded data control block modification routine will find the DCBD macro instruction very convenient for referencing the fields of the control block.

Source 2 - The DCB Macro Instruction

Information may be supplied to the data control block by specifying operands in the DCB macro instruction. In this case, the DCB macro instruction, in addition to creating a data control block, also fills the specified fields with the attributes indicated via the operands.

Source 3 - The Catalog - At the time a data set is cataloged certain attributes (data set organization, data set disposition, device class, and data set affinity) are recorded in the catalog. When a user desires to re-open that data set for additional processing, information previously recorded in the catalog need not be specified again by another attribute source. If such recorded information is specified again by another attribute source, the previously recorded attribute information will take precedence.

Source 4 and 5 - The DDEF Macro Instruction (4) Or Command (5)

The DDEF macro instruction or command can supply the same information to all fields in the DCB as can be specified via the DCB macro instruction, except for the EODAD, SYNAD, and EXLST parameters. The DDEF macro instruction or command must be used for each data set to be processed because it is the only source of DSNAME, the data set name. The primary difference between the DDEF macro and command is the ability of the DDEF command to provide attribute information from the terminal at execution time rather than at assembly time.

Source 6 - Data Set Labels or Data Set Control Blocks (DSCB's)

At the time a data set is recorded on a storage device, a data set label or DSCB is created. The label or DSCB of an existing data set contains some data control block information. If fields in the data control block are still unspecified at open time, the information is taken from the data set label or DSCB and placed into the Data Control Block.

Priority of Sources

Many of the attributes of a data set, required by the system can be furnished from more than one of the six possible sources. In such cases, each of the sources providing this information is assigned a priority and the system will use the information from the source with the highest priority. When two or more of the sources have corresponding entries, the attributes in the lower priority sources will be ignored.

This priority scheme provides great flexibility since information omitted in a higher priority source can be supplied by a lower priority source. Thus, if attribute parameters such as DSORG are not specified in the higher level DCB and DDEF macro instructions they may be supplied dynamically, at the terminal, by the lower priority DDEF command, or by the DSCB or tape label.

If a field has been specified in the higher priority DCB or DDEF macro instructions at assembly time or by the user's program prior to OPEN it will not be possible to modify that field dynamically from the terminal (e.g., if there were a LRECL parameter specification in the DDEF command at the terminal and the DCB also contained an LRECL specification at assembly time, the LRECL specification of the DDEF command would be ignored. In many cases, if a lower priority source provides the same attribute data as a higher priority source but the data provided differs in each source, the system will issue diagnostics indicating this. The system will either assume the higher priority source contains the valid data and continue processing based on that source or it will require the user to issue the proper matching attribute data in the lower priority source. Thus, if a user specifies a data set's organization (DSORG) in a DDEF command for a data set that is already cataloged, it must agree with the DSORG recorded in the catalog or diagnostics will be issued asking the user to reenter the correct data set organization parameter or to default to the system default value. In the latter case, if the user fails to provide the proper information and does not use the system default option the system will abend the user's task.

Table 8. DCB Operands, Their Specification, Access Methods, and Alternate Sources (Part 1 of 2)

DCB Operand	Specifies	Applicable Access Method						Valid Alternate Sources			
		VSAM	VISAM	VPAM	BSAM	QSAM	IOREQ	User's Program	DEFINE DATA Command	Data Set Label	System Service Routines
DDNAME	Symbolic name identical to that used in ddname operand of DEFINE DATA command associated with data set	X	X	X	X	X	X	X			
DSORG	Data set organization	X	X	X	X	X	X	X	X		
RECFM	Record format information	X	X	X	X	X		X	X	X	X
LRECL	Logical record length	X	X	X	X	X		X	X	X	
EODAD	Address of user's end-of-data routine for input data sets	X	X	X	X	X		X			
SYNAD	Address of user's synchronous error exit routine (entered when an uncorrectable error occurs in I/O operation)		X	X*	X	X	X	X			
KEYLEN	Key length		X	X*	X			X	X	X	
RKP	Displacement of key from first byte of logical record		X	X				X	X	X	
PAD	Space to be left on each page of virtual index sequential data set (to allow subsequent insertions)		X	X*				X	X	X	
MACRF	Types of macro instructions used in processing data set (GET, PUT, READ, WRITE, etc.)				X	X		X	X		
DEVD	Device on which data set resides plus, for some device types, device-dependent information (data code, tape density, etc.)				X	X		X	some device dependent information	some device dependent information	

* (only for VISAM members)

(Continued)

Table 8. DCB Operands (Part 2 of 2)

DCB Operand	Specifies	Applicable Access Method						Valid Alternate Sources			
		VSAM	VISAM	VPAM	BSAM	QSAM	IOREQ	User's Program	DEFINE DATA Command	Data Set Label	System Service Routines
OPTCD	Optional service desired, write with validity check (for direct-access devices only)				X	X		X	X	X	
BLKSIZE	Maximum block length				X	X		X	X	X	
IMSK	Number code indicating what system error recovery procedures (if any) are to be invoked				X	X		X	X		X
EXLST	Address of user's exit list				X	X		X			
NCP	Number of consecutive READ, WRITE, or IOREQ macro instructions issued before, CHECK macro instructions.				X		X	X	X		X
BUFNO	Number of buffers				X			X	X		
BFALN	Buffer alignment				X						
BUFL	Buffer length				X			X	X		
EROPT						X					

The fields are presented in alphabetical order and are described in the following format:

```
NAME    (length)                              (name,name)
        specification of contents
```

NAME
is the keyword parameter name if the field may be supplied by keyword parameter in a DCB macro instruction. If the field is not supplied by keyword parameter, a meaningful name or phrase is given; e.g., retrieval address.

length
specifies the length of the field in bytes.

name
specifies the symbolic name or names which, when used in conjunction with the DCBD macro instruction, will address the data control block field.

An X in a bit position means that bit is not tested.

BLKSIZE (2-byte field) (DCBBLKSI,DCBBLK)
specifies a binary value for the maximum block length in bytes. The maximum value is 32,760.

BUFL (2-byte field) (DCBBUFL, DCBBUF)
contains a binary number that represents the length, in bytes, of each buffer obtained for a buffer pool. The maximum is 32,760.

BUFNO (1-byte field) (DCBBUFNO, DCBBUN)
contains a binary number that represents the number of buffers assigned to a data control block. The maximum is 255.

DDNAME (8-byte field) (DCBDDNAM, DCBDDN)
contains a name of up to eight characters

DEVD (1-byte field) (DCBDEVD, DCBDEV)

Code	Bit Pattern
DA	11000100
PT	11100111
TA	11100011
PR	11010111
RD	11011001
PC	11010101
no device specified	11010110

The additional keyword operands which are optionally used with DEVD= cause information to be inserted in device-dependent parameters 1 and 2.

Device Dependent Parameter 1 (1-byte field) (DCBDD1)

This byte is used to contain information from the operands that are subordinate to the DEVD= operand of the DCB macro instruction. It may contain the information for KEYLEN, DEN, STACK, PRTSP.

KEYLEN (DCBKEYLE, DCBKEY)
contains a binary number that represents the length, in bytes, of the key associated with a physical record. The maximum is 255.

DEN (DCBDEN)

Code	Density	Bit Pattern
0	200	00000011
1	556	01000011
2	800	10000011

When this field is used for DEN, it must not be altered during or after OPEN time.

STACK (DCBSTACK, DCBSTA)

	Bit Pattern
stacker select 1	XXXX0001
stacker select 2	XXXX0010

PRTSP (DCBPRTSP, DCBPRT)

Code	Bit Pattern
0 - no spacing	00000001
1 - space one line	00001001
2 - space two lines	00010001
3 - space three lines	00011001

Device Dependent Parameter 2 (1-byte field) (DCBDD2)

This byte is used to contain information from the operands that are subordinate to the DEVD= operand of the DCB macro instruction. It may contain information for either TRTCH or MODE.

TRTCH (DCBTRT)

Code	Bit Pattern
C	00100011
E	00111011
T	00010011
ET	00101011
Odd parity, no translation	00110011

MODE (DCBMOD)

Code	Bit Pattern
C	1000XXXX
E	0100XXXX

DSORG (2-byte field) (DCBDSORG, DCBDSO)

	Bit Pattern	
PS	0100000X	00000000
PSU	01000001	00000000
VI	01110001	00000000
VS	01110010	00000000
VIP	01110011	00000000
VSP	01110100	00000000
VP	01110101	00000000

EODAD (8-byte field) (DCBEODVD, DCBEOV) for V-con
(DCBEODRD, DCBEOR) for R-con
contains the address of the user's EODAD routine. The first word contains the entry point address. The second word contains the address of the PSECT for the EODAD routine. If the EODAD routine has no PSECT, the second word contains the address of the CSECT containing the EODAD routine.

EROPT (1-byte field) (DCBEROPT, DCBERO)

Code	Bit Pattern
ACC	10000000
SKP	01000000
ABE	00100000

Exceptional Condition Field 1 (1-byte field) (DCBEX1)

Error Caused By	Bit Pattern
GET	00000000
PUT	00000100
SETL	00001000
READ	00001100
WRITE	00001111
DELREC	00010100

Exceptional Condition Field 2 (1-byte field) (DCBEX2)

Type of Error	Bit Pattern
Keys equal - sequence error	00000100
Key not found	00001000
Keys out of sequence	00001100
Keys do not coincide	00001111
Keys coincide	00010100
Invalid retrieval address	00011000
Invalid record length	00011100
Position past end of data set	00011111
Position before beginning of data set	00100100
Exceed maximum number of overflow pages	00101000
Exceed maximum size of shared data set	00101100

EXLST (4-byte field) (DCBEXLST, DCBEXL)
contains the address of a user-supplied exit list. The exit list must be in the same CSECT as the data control block.

IMSK (4-byte field) (DCBIMSK,DCBIMK)
contains the system error mask. The bit pattern is as specified under IMSK in the DCB macro instruction.

LRECL (4-byte field) (DCBLRECL, DCBLRE)
specifies for format-F records the length in bytes of a logical record. For BSAM or QSAM the maximum value is 32,760 bytes; for VSAM, 1,048,576 bytes; for VISAM, 4,000 bytes.

MACRF (2-byte field) (DCBMACRF, DCBMAC)

	Bit Pattern
G	01000000 00000000
GS	01000001 00000000
GC	01000010 00000000
GSC	01000011 00000000
P	00000000 01000000
PS	00000000 01000001
PC	00000000 01000010
PSC	00000000 01000011
R	00100000 00000000
RC	00100010 00000000
RP	00100100 00000000
W	00000000 00100000
WC	00000000 00100010
WP	00000000 00100100

Note: For G[S], P[S] the bit pattern becomes the appropriate combination of the above bit patterns.

For R[C|P], W[|P] the bit pattern become the appropriate combination of the above bit patterns.

NCP (1-byte field) (DCBNCP)
contains a binary number that represents the number of consecutive READ or WRITE macro instructions that are to be issued before a CHECK macro instruction is given. The maximum is 99.

OPTCD (1-byte field) (DCBOPTCD, DCBOPT)

	Bit Pattern
W - write validity check	10000000
default condition, no validity check	00000000

OPTIONS (1-byte field) (DCBOPI)
contains the bit patterns specified by option parameters in the OPEN and CLOSE macro instructions.

OPEN OPT1	OPEN OPT2	CLOSE OPT	Bit Pattern
INPUT			XX0000XX
OUTPUT			XX1111XX
INOUT			XX0011XX
OUTIN			XX0111XX
RDBACK			XX0001XX
UPDAT			XX0100XX
	REREAD		01XXXXXX
	LEAVE		11XXXXXX
		REREAD	XXXXXX01
		LEAVE	XXXXXX11

PAD (1-byte field) (DCBPAD)
contains a binary number that represents the space, as percentage, left available within the pages of a VISAM data set, providing for insertions within the pages of a VISAM data set. The maximum is 50 (50 percent).

RECFM (1-byte field) (DCBRECFM,DCBREC)

	Bit Pattern
F - fixed	10XXXXXX
V - variable	01XX0XXX
U - undefined	11X00XXX
T - track overflow	XX1XXXXX
B - blocked	XXX1XXXX
S - standard	10XX1XXX
A - USASI control character	XXXXX10X
M - machine code control character	XXXXX01X
no control character	XXXXX00X
KEYLEN specified in Data Control Block	XXXXXXX1

EXAMPLE: If this byte contains 10010100, the record format is fixed length, blocked records with an ASA control character.

Retrieval Address for Virtual Access Method (4-byte field) (DCBLPDA,DCBLPA)

This field contains a retrieval address that is used for recording and repositioning to specified records of a data set.

Retrieval Address for QSAM (6-byte field) (DCBLPDQ)

This field contains a retrieval address that is used for recording and repositioning to specified records of a data set.

RKP (2-byte field) (DCBRKP)
contains a binary number that represents the displacement of the key field of a record from the first byte of the record.

SYNAD (8-byte field) (DCBSYNVD, DCBSYV) for V-con
(DCBSYNRD, DCBSYR) for R-con
contains the address of the user's SYNAD routine. The first four bytes contain the entry point address. The second four bytes contain the address of the PSECT for the SYNAD routine. If the SYNAD routine has no PSECT, the second word contains the address of the CSECT containing the routine.

APPENDIX G: DETAILED DESCRIPTION OF DDEF MACRO INSTRUCTION

This appendix describes the DDEF macro instruction as used to define any private data set or any nonstandard public data set. To define standard data sets refer to the description of the DDEF macro instruction. (Standard data sets have virtual sequential organization, are on direct-access public storage, and are arranged in units of pages.) Table 9 lists required and optional operand fields of the DDEF macro instruction for various types of data sets. The complete format of DDEF is as follows:

Name	Operation	Operand
[symbol]	DDEF	oplist-{text / addr}

oplist
specifies the list of operands supplied for the DDEF macro instruction as shown in Table 9.

ddname
specifies the symbolic data definition name associated with this data set definition. It provides the link between the data control block in the user's program and the data set definition. It must contain one to eight alphameric characters, the first of which must be alphabetic. The user is not allowed to use a ddname that begins with SYS; the system-reserved ddnames are prefixed with these characters.

PCSOUT
specifies that the program checkout subsystem is being used and a data set is being defined for dumps. One PCSOUT-type DDEF command or macro instruction is required when the DUMP command is to be employed.

dsorg
specifies a two-character code that indicates the organization of the data set. The codes are:

PS	SAM
VI	VAM index sequential
VS	VAM sequential
VP	VAM partitioned
RX	IOREQ

Default: The data set is assigned the type of organization specified at system generation time.

DSNAME
specifies the name of the data set as one of the following:

name
specifies the name of the data set. This is the dsname under which the data set may be cataloged or referred to during the task. A relative generation number and/or a partitioned data set member name may be included with the dsname.

This operand can be specified as the fully qualified name of a partitioned or nonpartitioned data set, a member of a partitioned data set, or a partitioned or nonpartitioned generation of a generation data group (identified by absolute generation name or relative generation number).

Table 9. Operands for DDEF Macro Instruction

```
┌──────────────────────────────────────────────────────────────────────┐
│                              Oplist                                  │
├──────────────────────────────────────────────────────────────────────┤
│ {ddname-symbol}                                                      │
│ {PCSOUT       }                                                      │
│ [,dsorg-{PS}]                                                        │
│         {VI}                                                         │
│         {VS}                                                         │
│         {VP}                                                         │
│         {RX}                                                         │
│ ,DSNAME={name  }                                                     │
│         {*name }                                                     │
│ [,DCB=(  [*ddname] [,DSORG=code] [,MACRF=code] [,BUFL=absexp]        │
│                    [,DEVD=code] [,BUFNO=absexp] [,BFTEK=S]           │
│                    [,NCP=absexp] [,RECFM=code] [,OPTCD=W]            │
│                    [,LRECL=absexp] [,BLKSIZE=integer]                │
│                    [,KEYLEN=absexp] [,PRTSP=integer]                 │
│                    [,STACK=absexp] [,DEN=integer] [,CODE=code]       │
│                    [,MODE=code] [,TRTCH=code] [,EROPT=code]          │
│                    [,PAD=absexp] [,RKP=integer] [,IMSK=code]  )]     │
│ [,UNIT=({DA [,datype-integer]  })]                                   │
│        {TA [,tatype-{7|7DC|9}]}                                      │
│        {AFF=symbol            }                                      │
│ [,SPACE=({TRK                  },primary-integer[,secondary-integer] [,HOLD])]
│         {CYL                   }                                     │
│         {reclength-integer     }                                     │
│ [,VOLUME=({PRIVATE                                    })]            │
│          {[volseqno-integer],[volserno-alphnum,...]   }              │
│ [,LABEL=([filseqno-integer] [,labeltype-{NL }] [,RETPD=days])]       │
│                                         {SL }                        │
│                                         {SUL}                        │
│ [,DISP=status]                                                       │
│ [,OPTION={CONC|JOBLIB}]                                              │
├──────────────────────────────────────────────────────────────────────┤
│ Note: absexp may be indicated for DCB subparameters in the DDEF      │
│ macro instruction, but integer must be specified if the DDEF command is
│ used.                                                                │
└──────────────────────────────────────────────────────────────────────┘
```

*name
: specifies the dsname, here prefixed by an asterisk (*), of a data set created under the IBM System/360 Operating System. Subsequent references to this data set name do not include the asterisk prefix. The *dsname may have a maximum of 44 characters.

DCB
: specifies the data control block information, as follows:

*ddname
specifies the data definition name of a previously issued DDEF command or macro instruction. The previous ddname is prefixed by an asterisk (*) to indicate that the data control block field of that DDEF is to be duplicated for the current DDEF macro instruction or command. Any new subparameters given in the remainder of the field take precedence over the corresponding subparameters of the previous DDEF command.

DCB Subparameters
Detailed descriptions of the data control block subparameters are given in the discussion of the DCB macro instruction for each access method, and in Appendix F.

Note: If the data set is or will be on tape, the DEN subparameter must be furnished to specify tape density. The only exception to this rule is the case when the tape conforms to the DEN default value, which is a value set at system generation time.

UNIT
specifies the type of device needed for the data set. Allowable devices are specified at system generation time and, therefore, may be changed. Direct-access devices may be specified for either public or private volumes. The other types of devices (tape) and unit affinity may be specified for private volumes only.

DA
specifies that a direct-access device is required for the data set.

datype
specifies the type of direct-access device as a four-digit number.

Default: The system selects the type of direct-access device, as specified at system generation time.

TA
specifies that a tape unit is required for the data set.

tatype
specifies the type of tape required. It may be one of the following:

7 - seven-track tape
7DC - seven-track tape with data conversion
9 - nine-track tape

Default: The system selects the type of tape, as specified at system generation time.

AFF
specifies unit affinity for SAM data sets only. The data set being defined is to be assigned the same device reserved for the data set identified by ddname, which is the data definition name of a previously issued DDEF command or macro instruction. This subfield is unacceptable if the data set is new and is to reside on a direct-access device.

SPACE
specifies the direct-access storage allocation for the data set. If the entire space field is defaulted, the direct-access storage allocation specified at system generation time is assigned.

TRK
specifies that the space requirements are expressed as number of tracks.

CYL
specifies that the space requirements are expressed as number of cylinders.

reclength
specifies the average record length, in bytes, of the physical records. It must be a decimal number not exceeding 32,767.

Default: If the data set organization is SAM, the unit of allocation is assumed to be a cylinder. If the data set organization is VAM, the unit of allocation is assumed to be a page (4096 bytes).

primary
specifies the number of units to be allocated to the data set; consists of a one- to three-digit decimal number.

Default: The primary space allocation assigned at system generation time is assigned.

secondary
specifies the number of units to be allocated each time the space allocated to the data set has been exhausted and more data is to be written; consists of a one- to three-digit decimal number.

Default: The secondary space allocation specified at system generation time is assigned.

HOLD
specifies that the unused storage assigned to this data set is not to be released when the data set is closed.

Default: Unused storage will be released.

VOLUME
specifies the volume on which the data set resides. Normally this field is used for an uncataloged data set that resides on a private volume. The entire field may be defaulted if a new data set is to be created on a public volume or if an old, cataloged data set is being defined.

PRIVATE
specifies that volumes are to be allocated from the system pool (i.e., the scratch tapes or disks available to the system operator). Once assigned, the volume remains the user's, exclusively, until he notifies the system operator that it can be returned to the pool.

volseqno
specifies the sequence number of the first volume of the data set to be read or written; consists of a one- to four-digit number. It is meaningful only if the data set has SAM organization, is cataloged, and its earlier volumes are not to be processed.

volserno
specifies the volume serial numbers identifying the volumes on which the data set resides. Each one must contain one to six alphameric characters. It is required for old uncataloged data sets that reside on private volumes; it may be supplied for new data sets that will reside on private volumes.

Default: If volseqno was specified, the data set is cataloged and the serial numbers will be retrieved from the catalog. If PRIVATE was specified, the system assigns a volume serial number. LABEL specifies the labeling conventions. If the entire label field is defaulted, the labeling conventions specified at system generation time are assigned. However, if the data set is cataloged, label information is retrieved from the catalog.

filseqno
specifies the file sequence number of a data set when multiple data sets are on one tape volume; consists of a one- or two-digit decimal number.

Default: The data set is assumed to be the first (or only) one on the tape volume.

labeltype
specifies either the type of labeling desired or the absence of labels. It may be one of the following:

NL - no labels
SL - standard labels
SUL - standard labels and user labels

Default: The system assumes the label type specified at system generation time.

RETPD
specifies the retention period of the data set, where days is a four-digit decimal number that indicates the time period, in days, that the data set is to be retained after its creation. Applicable for data sets on direct-access volumes or on labeled tapes.

Default: The retention period is assumed to be zero days, thus allowing immediate rewriting.

DISP
specifies the status of the data set. If DISP is defaulted in a DDEF for an existing cataloged public data set, the system will assume a value of OLD. If DISP is defaulted for any data set which does not yet exist, the system will assume a default value of NEW. It should be noted that for existing uncataloged private data sets the DISP value must be explicitly specified as OLD. If the user tries to default such a data set a DISP value of NEW is assumed and causes a system error. The various defaults and options are summarized below:

NEW - for a new data set.
OLD - for an old data set.
MOD - the data set exists but is being added to. MOD causes logical positioning after the last record of the data set. It applies only to SAM data sets on private volumes.
Defaults - OLD - for old cataloged data sets.
NEW - for a new data set or for an old uncataloged private data set.

OPTION
specifies that either a job library is being defined or a data set is being added to the concatenated data set named as ddname.

JOBLIB
specifies that the data set is to be used as a job library. The data set name specified in the dsname field will be entered into the program library list.

CONC
specifies the concatenation of this data set with one or more data sets whose data definitions have the same ddname. Only input data sets that are not job libraries can be concatenated. The order of concatenated data sets is the same as the order in which they are defined.

The DDEF macro instruction or command that defines any cataloged data set is brief and simple. The only required operand fields are ddname, dsname, and disp (disposition). Other operand fields are unnecessary since the organization of the data set is described in its catalog entry.

DDEF macro instructions or commands that define uncataloged data sets may be divided into two groups: those defining new data sets (i.e., data sets that will be generated during the run but do not exist as yet) and those defining old (already existing) data sets. These old uncataloged data sets can exist only on private volumes.

To define a new data set that will be written on a public volume, the user may use the ddname, dsname, space, dsorg, and label operand fields. Exactly which fields he uses other than ddname and dsname, which are required, depends on the character of his particular data set.

To define a new data set that will be written on a private volume, the user must give ddname, dsname, unit, and volume operands. If desired, he may also furnish dsorg, space, label, and disp fields.

The user defines an old, uncataloged data set just as it stands on his private volume. To do so, he must use the ddname, dsname, volume, unit, and disp fields. He may also employ the dsorg and label fields.

Note: The dcb field is required to specify tape density for any data set on tape. However, it may be defaulted if the tape density matches that established at system generation time.

The DDEF macro instruction or command also has several special uses:

1. To define a job library. Operand fields are as follows:

   ddname, VP, DSNAME=dsname, DISP=(OLD), OPTION=JOBLIB

   No other fields are required.

2. To define a data set for dumps. Operand fields are:

   PCSOUT, VI, DSNAME=dsname

   Other fields are as needed.

3. To complete the data control block of a data set at execution time. The dcb field is included in this case; other operand fields are as needed for the particular data set.

4. To concatenate data sets (i.e., to define them, for input purposes only, so that several data sets can be read as if they formed a single data set. The OPTION=CONC field is included; other fields are as needed for each data set. The OPTION=CONC field must be given in the DDEF for each data set except the first-defined member of the concatenation. The remaining data sets in the concatenation must each have the same ddname as the first-defined data set.

The DDEF macro instruction or command causes a system entry to be established for the DDEF information so that allocation routines and access methods can refer to it. The link between this information and the problem program's reference to the data set (i.e., the data control block) is the data definition name. The entry containing the DDEF information is maintained until the user logs off or until, through the RELEASE macro instruction or command, the data set is released.

The DDEF macro instruction or command also results in a request, when necessary, for device allocation and volume mounting if the defined data set is private and resides on a demountable volume such as a reel of tape or a disk pack.

Typical Use of DDEF Operand Fields

Case	ddname	dsorg	dsname	dcb	unit	space	volume	label	disp	option
Read a cataloged data set	x		x						x	
Read an uncataloged data set	x	[x]	x		x		x	[x]	x	
Write a data set on a public volume	x	[x]	x			[x]		[x]	[x]	
Write a data set on a private volume	x	[x]	x		x	[x]	x	[x]	[x]	
Modify any data set on a private volume	x	[x]	x		x		x	[x]	x	
Concatenate cataloged data sets while reading private volumes (for each concatenated data set except first in concatenation)	x	[x]	x						x	x
Key: [] indicates operand entry is optional.										

Data Set Organization Requirements

Data Set	Data Set Organization (dsorg)				Comments
	PS	VS	VI	VP	
Any data set on a public volume		x	x	x	
Any data set on a private volume	x	x	x	x	PS applies tc direct-access and tape volumes; VS, VI, and VP apply only to volumes on direct-access devices
Any member of a partitioned data set		x	x		The same partitioned data set may include both VS and VI members. (The member must be either VS or VI.)
SYSIN data set		x	x		
Language Processing Source data set for language processing			x		Line data set only. If source data sets are entered from terminal, a line data set is automatically built
Source statements stored as part of SYSIN data set		x	x		A line data set will be built from source statements
Object module produced by language processor		x			The object module automatically becomes a member of the most recently defined job library, if any, or of the user's library (SYSULIB).
Job library				x	
Listing data set produced by language processor			x		
Input/Output PCSOUT data set			x		
Input to WRITE TAPE		x	x		
Input to PRINT	x	x	x		
Input to PUNCH		x	x		
Special Command Usage Data set for CALL DATA DEFINITION			x		Line data set only
Data set for LINE?			x		Line or language processor listing data set cnly
Data set created by DATA		x	x		User option. If VI, must be line data set
Data set created by MODIFY			x		User option determines whether VI is line data set or not

PROGRAMMING NOTES: The DDEF macro instruction or command may be used in conversational and nonconversational tasks.

The user's replies to diagnostic messages issued for his DDEF macro instruction or command should be guided by:

1. If the diagnostic message calls for reentering an element within a given operand field, only that element should be reentered. Preceding and/or following delimiters are unnecessary. Default is acceptable.

2. If the diagnostic message calls for reentering a complex operand field, the whole field should be reentered, including keyword and equal sign. Default is acceptable.

3. If the diagnostic message calls for reentering an operand field that consists of only one element in addition to the keyword, the reply may be either the element alone or the keyword, equal sign, and element.

4. If the diagnostic message calls attention to an inconsistency and asks the user to (re)enter one of two or three specified operands, the reply must be a complete operand field. A default is acceptable only if so stated in the message.

The user is informed if the DDEF macro instruction or command cannot be completed. This action can occur for one of these reasons:

1. Invalid punctuation in the operand string.
2. User's volumes cannot be mounted.
3. Sufficient space cannot be allocated.
4. More than three logical inconsistencies were detected in the DDEF macro instruction or command.

Whenever possible, correction and completion of the command will be attempted. But if diagnostic messages indicate that a parameter was misunderstood because of a punctuation error in the operand string, the user should interrupt the operation (by pressing the ATTENTION key) and reenter the corrected command. In confirmation mode, he may prefer to wait for prompting.

The user must never reenter a parameter or part of a parameter that was not requested.

If a keyword is missing or invalid, the pertinent elements following it must be reentered after the corrected keyword and equal sign are typed.

If a parameter occurs twice in the operand string, the second occurrence is preferred. All elements belonging in the earlier occurrence are erased.

DDEF prompting messages are issued according to the operand information already supplied. Unnecessary prompting is kept to a minimum.

If the user's program is being executed in conversational mode and an undefined ddname is referenced, prompting messages for DDEF operands will be issued to the user regardless of confirmation mode.

At completion of execution of the DDEF macro instruction or command, a code is loaded into the low-order byte of general register 15. The significance of these codes is:

Code	Significance
00	No error
04	Undefined dsname (for old data set)
08	Multi-defined dsname (for new data set)
0C	Attention interruption
10	Dsorg in DDEF parameter list is not the same as dsorg in catalog
20	Space cannot be allocated
40	ddname not unique
80	Any error condition not listed above

L- AND E-FORM USE: The oplist operand is required in the L-form and is not permitted in the E-form of this macro instruction. Only the text form of the operand may be used in the L-form of this macro instruction.

APPENDIX H: MACRO INSTRUCTION GENERATION OF LITERALS

The TSS/360 assembler places literals in a module's first declared PSECT if one has been declared. If no PSECT has been declared, address literals are treated as any other literals; i.e., placed in whatever literal pool is proper. Table 10 indicates which macro instructions may generate literals, which operands of each macro instruction are involved, and under what conditions the literals are generated.

The table is arranged alphabetically by macro instruction name and includes all macro instructions, whether or not they may generate literals. The table also indicates whether a literal is generated as an address literal or as some other kind of literal.

From the table a user can, therefore, determine if the expansion of a particular macro instruction will generate a literal. If a literal is generated, the user must be sure that the location containing the literal is covered by a base register at the time the macro instruction is executed.

Privileged users must establish save-area cover in register 13 before executing any macro instruction that generates a type I linkage. The macro instructions shown in Table 10 are flagged with an asterisk if they generate a type I linkage in an assembly module that is declared to be privileged by means of the DCLASS macro instruction.

Table 10 includes all TSS/360 macro instructions whether they are documented in this publication or in System Programmer's Guide.

Table 10. Literals Generated by Macro Instructions (Part 1 of 6)

Macro Name	Source of Literal			Condition Under Which Literal is Generated					
	Operand	Linkage	Adcon Literal	Operand >$2^{12}-1$	E-form Only	Operand is Specified and Not R.N.	Program is Privileged	Standard-form Only	Operand not Text
ABEND*		X	X				X		
ADCON									
ADCOND									
AETD									
ARM									
BPKD									
BSP*		X	X				X		
CALL									
CAT*		X	X				X		
	oplist		X			X		X	X
CDD*		X	X				X		
	oplist		X			X		X	X
CDS*		X	X				X		
	oplist		X			X		X	X
CHECK*		X	X				X		
CLATT									

(Continued)

Table 10. Literals Generated by Macro Instructions (Part 2 of 6)

CLOSE*		X	X				X		
CNTRL*		X	X				X		
	number					X			
COMMAND*		X	X				X		
CSTORE*		X	X				X		
	lng			X	X				
	atr			X	X				
DCB									
DCBD									
DCLASS									
DDEF*		X	X				X		
	oplist		X			X		X	X
DEL*		X	X				X		
	dsname		X			X		X	X
DELETE									
DELREC*		X	X				X		
DIR*		X	X				X		
DQDECE									
EBCDTIME		X	X						
ESETL*		X	X				X		
EXIT									
FEOV*		X	X				X		

(Continued)

Table 10. Literals Generated by Macro Instructions (Part 3 of 6)

FIND*		X	X				X		
	length			X	X				
FREEBUF*		X	X				X		
FREEMAIN*		X	X				X		
	LV			X					
	VAR					X			
FREEPOOL*		X	X				X		
GATRD*		X	X				X		
GATWR*		X	X				X		
GDV									
GET									
GETBUF*		X	X				X		
GETMAIN*		X	X				X		
	VAR					X			
	PACK					X			
	PR					X			
	LV			X					
GETPOOL*		X	X				X		
	number					X			
	length					X			
GTWAR*		X	X				X		
GTWSR*		X	X				X		
INTINQ*		X	X				X		
	MODE null or R	X	X						
LOAD									

(Continued)

Table 10. Literals Generated by Macro Instructions (Part 4 of 6)

MCAST									
MSGWR*		X	X				X		
NOTE*		X	X				X		
OBEY									
OPEN*		X	X				X		
PAUSE*		X	X				X		
POINT*		X	X				X		
PR*		X	X				X		
	oplist		X			X		X	X
PRMPT									
PRTOV									
PU*		X	X				X		
	oplist		X			X		X	X
PUT									
PUTX									
RAE	area		X			X			
READ*		X	X				X		
	length			X	X				
REL*		X	X				X		
	oplist		X			X		X	X
RELEX*		X	X				X		
RELSE*		X	X				X		

(Continued)

Table 10. Literals Generated by Macro Instructions (Part 5 of 6)

RETURN									
SAEC	When PFKMSK is a sublist				X	X			
	EP		X		X	X			
SAI	area		X			X			
SAVE									
SEEC	MSGITH				X	X			
	INTTYP				X	X			
	EP		X		X	X			
SETL									
SIEC	EP		X		X	X			
SIR*		X	X				X		
SPEC	When INTTYP is a sublist				X	X			
	EP		X		X	X			
SSEC	EP		X		X	X			
STEC	INTTYP				X	X			
	EP				X	X			
STIMER									
STOW*		X	X				X		
SYSIN									
TRUNC		X	X				X		
TTIMER									
USAIT									
WRITE*		X	X				X		
	length			X	X				
WT*		X	X				X		
	oplist		X			X		X	X
WTL*		X	X				X		
WTO*		X	X				X		
WTOR*		X	X				X		

APPENDIX I: INTERRUPTION HANDLING FACILITIES

Time Sharing System/360 provides macro instructions that permit the user to control task interruptions (Figure 3).

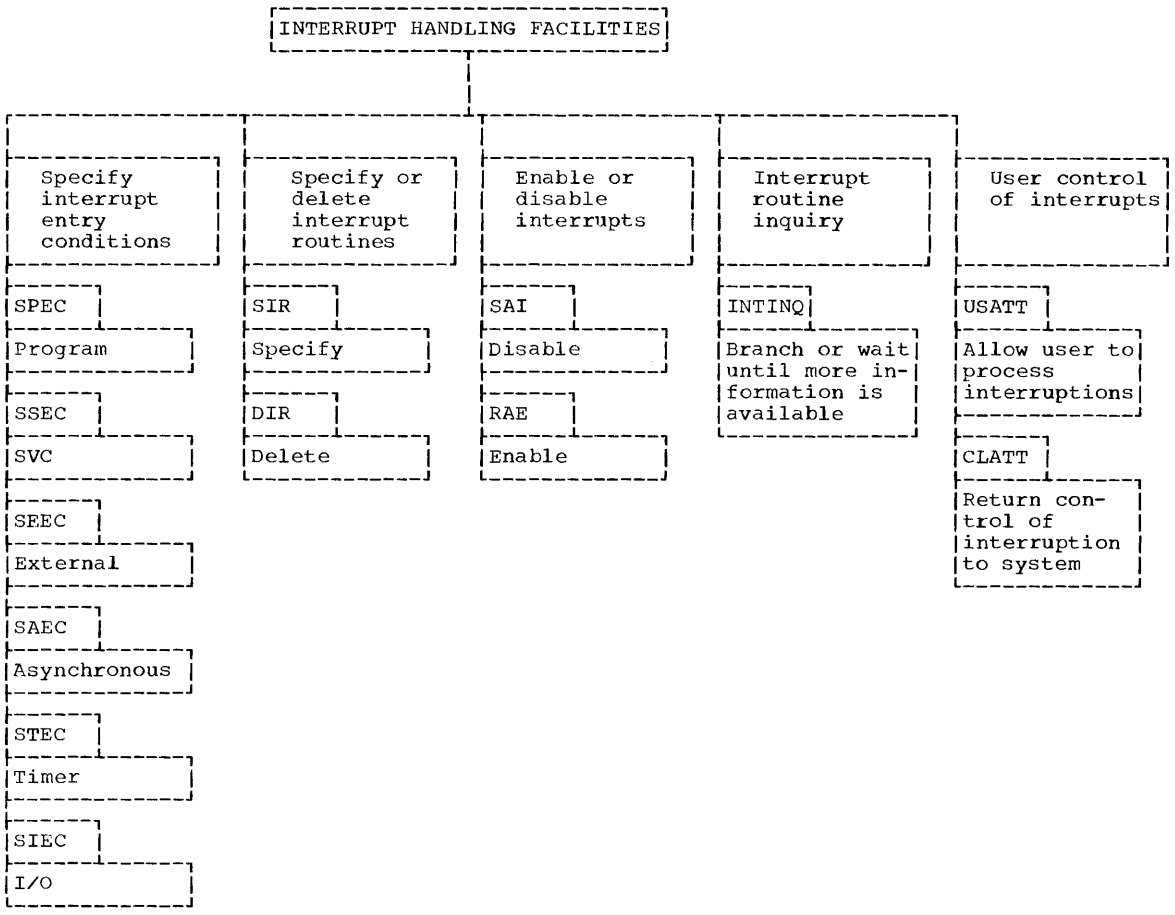


Figure 3. TSS/360 Interruption Handling Facilities

The six types of task interruptions and their corresponding macro instructions are:

Program	SPEC
Supervisor Call	SSEC
External	SEEC
Asynchronous	SAEC
Timer	STEC
Input/Output	SIEC

Interruption handling consists of responding to task interruptions. The most significant features of the interruption handling facility are priority interruption control and interruption delay.

The user must decide how to respond to each type of interruption, or he may elect to ignore certain interruptions. Interruptions may be serviced by one or more routines.

The following macro instructions create interrupt control blocks (ICBs) that specify what task interruptions are to be processed, under what conditions the user's interruption routine is to be entered, and the entry point address of the user's interruption routine.

Macro instruction	Function
SIR	Makes an interruption routine available for use, by establishing control references to it and also setting the priority of an interruption routine.
DIR	Deletes control references to a previously specified interruption routine.

To ensure that an interruption routine will not be interrupted, two macro instructions are provided:

Macro instruction	Function
SAI	Inhibits interruptions from taking place. However, no interruptions will be lost because they are queued up.
RAE	Enables interruptions to occur.

Interruptions are queued according to type and are dispatched according to priority. The following macro instruction provides flexibility within an interruption routine:

Macro instruction	Function
INTINQ	Allows an interruption routine to temporarily relinquish control, enter a wait state, branch conditionally, or delete all impending interruptions on a queue.

ESTABLISHING INTERRUPTION ROUTINES

Interruption routines are established through the SPEC, SSEC, SEEC, SAEC, STEC, and/or SIEC macro instructions. Control references to the routine and its priority are set by the SIR macro instruction. Any interruption routine may be made unavailable by the DIR macro instruction. Another SIR macro instruction will make the interruption routine available again.

PROCESSING AN INTERRUPTION

When an interruption occurs, an asynchronous exit is taken from the that occurs is made available in a communication area. Using this information, the interruption routine can perform any calculations necessary, issue input/output macro instructions, and do whatever is necessary to respond to the interruption. The INTINQ macro instruction may be issued in the interruption routine. Issuing a RETURN macro instruction causes control to be returned to the interrupted routine or to another queued interruption routine.

If interruptions are not disabled by an SAI macro instruction, interruptions of higher priority interrupt an interruption routine of lower priority.

COMMUNICATION AREA

The communication area, in addition to its primary purpose of holding interruption information for an interruption routine, allows information to be passed between the interruption routine and the interrupted program. A field in the communication area may be used as an event control block (ECB) where completion of interruption processing can be posted.

ENTRY

When an interruption routine is entered, register 1 contains the address of a two-word parameter list. The first word of the parameter list contains the address of a commmunication area, and the second word contains the address of a data control block (Figure 4).

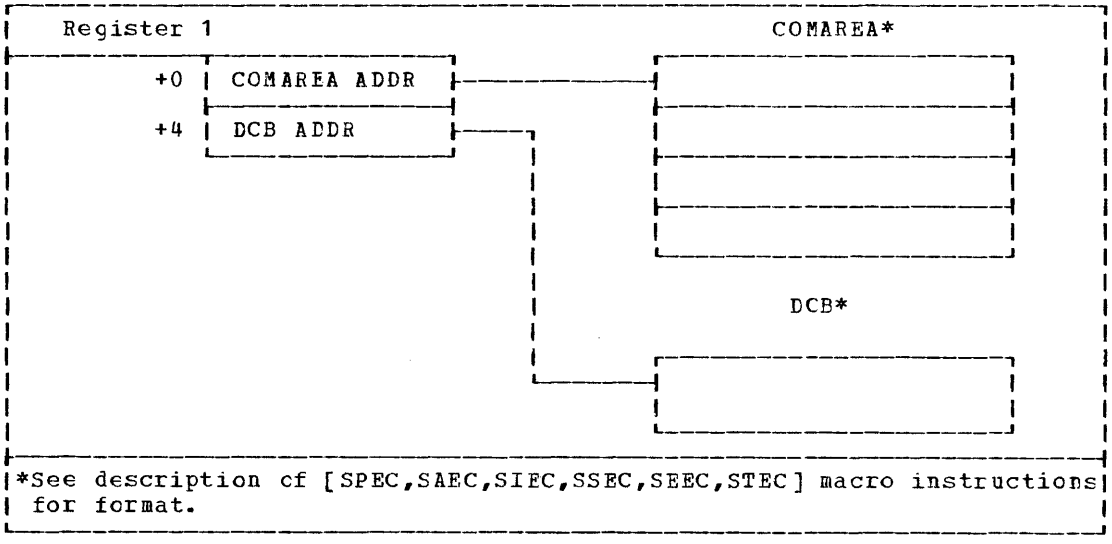


Figure 4. Information Available Upon Entry to an Interrupt Routine

APPENDIX J: THE TSS/360 SYSTEM MACRO AND COPY LIBRARY

A symbolic library is composed of a symbolic component and index component. The symbolic component may contain any collection of named groups of symbolic lines called parcels; thus, a collection of macro definitions corresponding to the TSS/360 system macro instructions, together with any parcels to be accessed by means of the COPY assembler instruction, form the symbolic component of the TSS/360 system macro and COPY library. This library provides the TSS/360 assembler with the macro definitions and COPY parcels it needs, when system macro instructions or COPY statements are encountered.

In this library, each macro definition is a group of symbolic lines whose name (parcel name) is the same as that of the operation of the definition's prototype and the corresponding macro instruction. Each COPY parcel is a group of symbolic lines to whose name a COPY statement must refer, to copy the parcel into a program. The symbolic component of the system macro and COPY library is normally cataloged as a virtual index sequential data set. The organization and format of this component is shown in Figure 5. The format of each symbolic line, shown in Figure 6, is that of a record in a line data set. The lines of information within the symbolic component are ordered by line number. The number of the first line of each parcel is used to index the symbolic component.

The index component is a table that relates the name of each parcel to the number of its first line. Thus, any parcel in the system macro and COPY library may be located within the symbolic component by matching the operation, of the corresponding macro instruction or operand of the corresponding COPY statement, to the appropriate entry in the index. The index component is normally cataloged as a virtual sequential data set. It consists of a single format-U record.

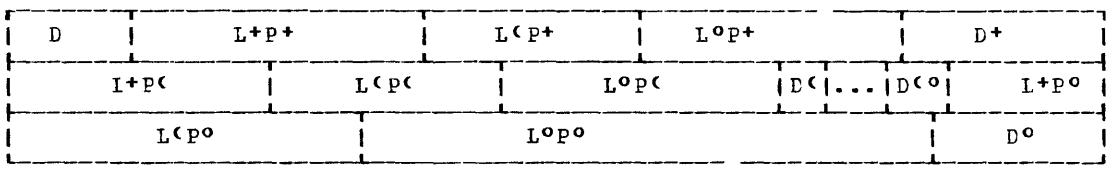


Figure 5. System Macro and COPY Library Symbolic Component Format

D
is a 21-byte line whose first character, always a right parenthesis, marks it as the delimiter line for the jth parcel. The 8-character field following the right parenthesis contains the name of the (j+1)th parcel, left-adjusted with trailing blanks.

L P
is the ith line of parcel j consisting of four more than the number of bytes given in its length field.

Note: The first line of a parcel is L+P, not D+. D', ... D'° are synonyms for the following parcel. Any parcel may have synonyms (aliases).

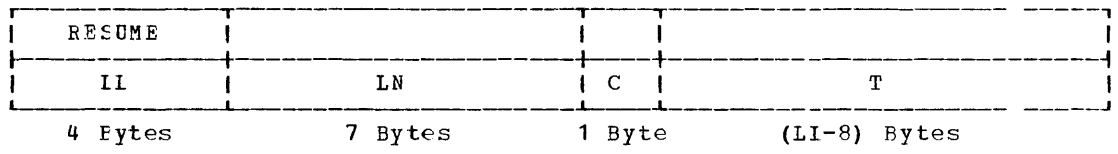


Figure 6. Format of a Line in a Line Data Set

LL
is the length of the line excluding the LL field.

C
is a code whose values and their meanings are:

Code	Meaning
01	The line originated at a terminal keyboard
00	The line was obtained as a card image

Note: C is normally 00 for all lines of the system macro and COPY library.

LN
is the line number.

T
is the text of the symbolic line consisting of LL minus eight characters.

SYSTEM MACRO AND COPY LIBRARY SERVICE FACILITIES

Generating the Library

The DATA command is used initially to create a line data set consisting of the desired collection of system macro definitions and assembler COPY parcels. This data set becomes the symbolic component of the system macro and COPY library when operated upon by the symbolic library indexing routine, SYSINDEX or the symbolic library index build routine, SYSXBLD. The MODIFY command may be used to change the symbolic component when necessary.

Changes are made as a function of line number. Each line in a line data set contains a line number; lines in the data set are ordered by line number. Once the line data set is created, RUN command is used to execute SYSINDEX. Alternatively, a user's program may perform the required function by calling SYSXBLD. These routines create the index (CHASLX) which relates the name of each parcel to its first line. When the MODIFY command is used to change the line number of the first line of any parcel in the symbolic component, an updated index must be created. The use of MODIFY does not otherwise require the subsequent use of SYSINDEX or SYSXBLD.

Using Symbolic Libraries

The TSS/360 assembler uses the symbolic library search routine (SYSEARCH) to locate a parcel in the system macro and COPY library when it encounters a system macro instruction or an assembler COPY statement. SYSEARCH inspects the index which the assembler has presented to it, and returns with a return code of 4 if the required parcel is not in the

library. If the required parcel is in the library, SYSEARCH returns with the number of the first line of the parcel and a return code of 0.

The assembler uses the line number obtained from SYSEARCH, in conjunction with a SETL macro instruction, to position the symbolic component at the required parcel. Successive statements are then obtained by using the VAM GET facility.

SYSEARCH is called to determine whether the parcel is present and, if so, positions the symbolic component to the designated parcel. If the parcel is not present, exit is made with a return code of 4; otherwise, it exits with a return code of 0. In the latter case, SYSEARCH is repeatedly called to obtain successive lines of the parcel.

As each line is obtained, SYSEARCH determines whether the line is still in the required parcel by testing the first text character. If that character is a right parenthesis, or if the EODAD sequence receives control, exit is made with a return code of 4; otherwise the line is presented to the assembler and exit is made with a return code of 0. When the assembler is retrieving a macro definition from the library, it will normally sense the end of the definition when it receives the definition's MEND statement.

If, instead, it detects a return code of 4 before it receives the MEND statement, it assumes that a library format-error exists. When the assembler is retrieving a COPY parcel, it relies upon a return code of 4 from SYSEARCH to detect the end of the parcel.

Requesting Symbolic Library Services

The symbolic library indexing routine (SYSINDEX) is a system utility routine that processes the user's input parameters. It is initiated by a command language RUN command. The input parameters expected by SYSINDEX are contained in one or more SYSIN control statements.

SYSINDEX expects these parameters to arrive in a control statement of the form:

keyword=parameter,keyword=parameter

LENGTH=integer
: specifies that each parcel name in CHASLX is composed of the number of characters given by integer.

HEADER=character
: specifies a single character which is compared with the first byte of each source line to determine whether that line requires an index entry. HEADER is not used if the user supplied SCAN (below).

SCAN=symbol
: specifies the name of a user-supplied subroutine which is called to inspect each successive line of the symbolic component. This routine will determine whether a given line requires an entry in CHASLX. SCAN is not used if the user supplies the HEADER parameter

The build symbolic library index routine (SYSXBLD) constructs the index portion (CHASLX) of the symbolic library. It is invoked by means of a CALL macro instruction of the following format:

Name	Operation	Operand
[symbol]	CALL	SYSXBLD,(length-addr,[header-addr][,scan-addr])

length
: specifies the location of the length of parcel names in the library.

header
: specifies the location of a character used in determining what lines of the symbolic component require index entries. The header character is compared with the first character of each line to make this determination. If header is given, it must be the second element of the sublist and scan must not be given.

scan
: specifies the location of an eight-character name of a user's scan routine. The name must be left-adjusted and filled with trailing blanks if necessary. The user's scan routine is called as each symbolic line is obtained to determine whether the line requires an index entry. If scan is given, it must be the third element of the sublist and header must not be given.

The symbolic library search routine (SYSEARCH), used to locate information stored in a symbolic library, is invoked by means of a CALL macro instruction of the following format:

Name	Operation	Operand
[symbol]	CALL	SYSEARCH,(index-addr,name-addr,linno-addr)

index
: is the address of the index component (CHASLX) of the symbolic library to be searched. CHASLX must be brought into storage by the user.

name
: is the address of the first byte of the name to be located. This name must be of the length specified to SYSINDEX or SYSXBLD during the creation of the index, and must be left-adjusted with trailing blanks.

linno
: is the location at which the SYSEARCH routine is to store the retrieval line number it obtains.

On exit, a code will be returned to the calling program in the return code register. The code will be one of the following values:

0 - if the name was located. The retrieval line number will be placed in the location designated by the third parameter.

4 - if the name could not be located.

APPENDIX K: SHARING VIRTUAL STORAGE DATA SETS

To be concurrently accessible to more than one task, a data set must have one of the following organizations:

- Virtual sequential
- Virtual index sequential
- Virtual partitioned

Physical sequential data sets cannot be used concurrently by more than one task.

To prevent several users from concurrently updating the same record of a virtual storage data set, interlocks are put on the data set while it is being used. The interlocks, read and write can be imposed at three levels: page, data set, or member.

Types of Interlocks

A read interlock is imposed to prevent other users from writing into a data set, member, or page of a data set. Multiple read interlocks may be established for a data set or member, permitting several users to read it simultaneously; or the interlocks may be set on a page basis, giving several users simultaneous access to the records within a page. A read interlock cannot be set if a write interlock has already been set for the data set or page.

A write interlock prevents any user, other than the user who set the interlock, from reading or writing into a data set or page. Only one write interlock can be set at a time; thus, once a write interlock is set, neither read nor write interlocks can be applied until the write interlock is reset.

Levels of Interlocks

- Data set interlock - set according to the OPEN option specified, as shown in Table 11. This level of interlock restricts the use of subsequent OPEN macro instructions on shared data sets. The interlock is reset when the data set is closed.
- Member interlock - set when the FIND macro instruction is issued for a member of a virtual partitioned data set. A member interlock is reset when a STOW type-R or CLOSE or FIND macro instruction is issued.
- Page interlock - set to ensure that the user has exclusive control of a record while he is processing it. A page-level interlock is reset when a reference is made to another page in the data set or when the data set is closed.

Table 11. Effect of OPEN Options on Data Set Interlocks

OPEN option	VSAM data set	VISAM data set	VPAM data set
INPUT	read interlock set	read interlock set	read interlock when FIND issued
OUTPUT	write interlock set	write interlock set	write interlock set when FIND issued
INOUT OUTIN UPDAT	write interlock set	read interlock set	when FIND is issued: write interlock is set for VSAM members; read interlock is set for VISAM members

User Considerations

The only way a user can gain exclusive control of a shared VISAM data set is to open it for OUTPUT. Although a data set is opened for OUTPUT, a user may actually only want to read the data set.

When updating a VISAM data set, the record to be updated should have been obtained by a READ (type KX). If users of a shared data set do not employ this procedure, two tasks may concurrently refer to the same page using either the GET or READ (type KY) macro instructions and decide that a record within the page is to be updated. Since both tasks use WRITEs to the same page, the task that issues the last WRITE macro instruction cancels the effects of the previously issued WRITE. The following sequence prevents this situation:

```
GET  (1)
.                              decision that updating of the record
.                              is required
.
READ DECB, KX, (1), (0), (2)
.
.                              update record
.
WRITE DECB, KS, (1), (0), (2)
```

A READ (type KZ) by retrieval address should not be employed by users of VISAM shared data sets since the retrieval address of the desired record can be shared by another task.

Coding sequences within a task may produce task looping that cannot be detected by the access method. Consider, for example, this sequence:

```
READ DECB, KX, (1), (0), (2)
GET (1)
```

where the READ and GET macro instructions refer to different DCBs within the same task. This situation produces a task loop, since the GET macro instruction waits for the write interlock, set by the previous READ macro instruction, to be reset. The write interlock will not be reset since it was set in the same task that is waiting for the write interlock to be reset. The user must pay close attention to the rules of interlock setting and resetting when dealing with multiple opened DCBs within a given task.

APPENDIX L: OPEN/CLOSE GENERATED PARAMETER LIST

One doubleword parameter list is generated for each data set DCB being opened or closed and placed in a table, as described below:

byte	contents
0-3	Address of the DCB
4	OPEN/CLOSE option code
5-7	(00 00 00) 16

The bit configurations for the option codes are indicated below.

bits 0-7	option
00XXXXXX	another DCB is to be opened or closed
10XXXXXX	this is the last DCB to be opened or closed
XX01XXXX	REREAD
XX11XXXX	LEAVE
XXXX0000	INPUT
XXXX1111	OUTPUT
XXXX0011	INOUT
XXXX0111	OUTIN
XXXX0001	RDBACK
XXXX0100	UPDAT

INDEX

| (see exclusive OR)
[ ] (see braces)
{} (see brackets) ... (see ellipses)
& (see ampersand)
( ) (see parentheses)

ABEND macro instruction 176
absexp, definition of 13
absolute expression, definition of 13
absolute generation name 15
accessing data sets 41
  VSAM 41,42-45
  VISAM 41,16-54
  VPAM 41,55-60
  BSAM 41,61-82
  QSAM 41,83-94
  IOREQ 41,95-100
access methods (see accessing data sets)
ADCON macro instruction 133,134
ADCOND macro instruction 133,137
addr, definition of 9,11
address constant
  (see ADCON macro instruction)
addrx, definition of 9,11
addx, definition of 9,11
AETD macro instruction 170
AFF (affinity operand in DDEF macro instruction) 329
allocation
  of direct access storage (see SPACE) of virtual storage 126
alphameric characters, definition of 16
alphnum, definition of 10
ampersand
  use in characters 14
  use in text 14
apostrophes
  use in characters 14
  use in oplist operands 16
  use in text 14
ARM macro instruction 138
  use with CALL 139
  use with DELETE 143
  use with LOAD 142
armed adcon group
  (see also ARM macro instruction)
ATPOL macro instruction 172
attention interrupt, control of 169,170
AWAIT macro instruction 213

backspace
  (see BSP macro instruction)
backward reading of magnetic tape 63
basic sequential access method
  (see BSAM)
BFALN (DCB operand) 26,32
BFTEK (DCB operand) 26,33
blanks
  use in characters 14
  use in text 14
BLKSIZE (DCB operand) 26,31,232
block size
  (see BLKSIZE)
BPKD macro instruction 208
braces, use of 8
brackets, use of 9
BSAM macro instructions 61
  READ 62
  WRITE 64
  CHECK 67
  DQDECB 69
  GETPOOL 72
  GETBUF 70
  FREEBUF 71
  FREEPOOL 73
  BSP 74
  CNTRL 75
  FEOV 77
  POINT 78
  NOTE 80
  PRTOV 81
BSP macro instruction 74
BUFCB (DCB operand) 26,33
buffer
  alignment of (see BFALN)
  length (see BUFL)
  pool 7372,73
buffering technique
  (see BFTEK)
BUFL (DCB operand) 26,33,232
BUFNO (DCB operand) 26,32,232
bulk output facilities 104
  PR 104
  PU 107
  WT 109

CALL adcon group 136
call data definition
  (see CDD macro instruction)
CALL macro instruction 139
capital letters, use of 9
CAT macro instruction 113
cataloging data sets 113
  CAT 113-116
  DEL 116-117
CDD macro instruction 34
CDS macro instruction 101-103
channel program
  (see NCP)
character and switch table 179
characters, definition of 14
  operand form 11,14
  value mnemonic 10
character translation table 179
CHECK macro instruction

for BSAM 67
for IOREQ 98
check protection class
(see CKCLS macro instruction)
CKCLS macro instruction 132
CLATT macro instruction 170
CLIC macro instruction 178
CLIP macro instruction 178
CLOSE macro instruction 118
parameter list 260
CLOSE (TYPE=T) macro instruction 118
CNTRL macro instruction
for BSAM 75
for QSAM 90
code definition of
operand form 11,13,14
value mnemonic 10,11
COMAREA
(see communication area)
command creation 208
BPKD 208
GDV 211
COMMAND macro instruction 174
command mode 173-177
communication with operator 197
WTO 197
WTC 197
WTOR 198
with system log 197
WTL 199
inter-task
(see VSEND and VSENDR macro instructions)
with SYSIN/SYSOUT 179
GATRD 179
GATWR 182
GTWAR 183
GTWSR 184
SYSIN 185
PRMPT 188
MSGWR 191
MCAST 193
communication area
for SAEC macro instruction 160
for SEEC 156
for SIEC 165
for SPEC 154
for SSEC 158
for STEC 163
CONC (DDEF macro instruction operand) 241
concatenating data sets
(see CONC)
connecting data sets 37
OPEN 37,38-40
control characters 224
(see also RECFM and MCAST)
control of attention interrupts
(see AETD, USATT, and CIATT macro instructions)
COPY library 256
copying data sets 101
CDS 101
create catalog entry
(see CAT macro instruction)
CSTORE macro instruction
CTT
(see character translation table)
CYL (DDEF operand) 240

Data control block 229
(see also DCB macro instruction)
data definition name
(see DDNAME)
data event control block
(see DECB)
data set
concatenating (see CONC)
defining attributes (see DCB and DDEF macro instructions)
interlock 258-259
name 14
in DDEF macro instruction (see DSNAME)
organization (see dsorg: DSCRG)
sharing 258-259
data set management 21
DCB (DDEF operand) 238,239
DCB macro instruction 22,25-34,229
DCBD macro instruction 22,35
DCLASS macro instruction 131
DDEF macro instruction 237-247
DDNAME (DCB operand) 26
ddname (DDEF operand) 24,237,238,239
DECB
with CHECK, BSAM 67,98
with IOREQ 98
with READ 48,62
decimal integer, definition of 13
define a data set 22
DDEF 22,23-25
DCB 22,25-34
CDD 22,34
DCBD 22,35
DEL macro instruction 116-117
DELET macro instruction 148
delete
catalog entry (see DEL macro instruction)
interrupt routine (see DIR macro instruction)
module (see DELETE macro instruction)
record (see DELREC macro instruction)
DELETE adcon group 136
DELETE macro instruction 143
DELREC macro instruction 52
DEN (DCB operand) 26,30,233
DIR macro instruction 165
disconnecting data sets 118
CLOSE 118-122
CLOSE (TYPE=T) 119
DISP (DDEF operand) 24,238,241
disposition of a data set
(see DISP)
DLINK macro instruction 148
DQDECB macro instruction 69
DSNAME (DDEF operand) 24,237,238
DSORG (DCB operand 26,37,234
dsorg (DDEF operand) 24,237,238

EBCDTIME macro instruction 204
ellipsis, use of 9

enabling interrupts
(see RAE macro instruction)
enter command mode 173
ENTER macro instruction 148
EODAD (DCB operand) 26,29,223
ERASE macro instruction 123
EROPT (DCB operand) 29,33,234
ESETL macro instruction 52
exclusive control of a record 259
exclusive OR, definition of 8
execute form macro instruction 19
exit list 215
(see also EODAD, SYNAD)
EXIT macro instruction 175
EXLST (DCB operand) 26,32,234
(see also exit list)
expansion of a macro instruction definition of 7
explicit
address, definition of 12
call 140
deletion 228
linkage 228

FEOV macro instruction 77
FIND macro instruction 55
FINDDS macro instruction 36
FINDJFCB macro instruction 36
FREEBUF macro instruction 71
FREEMAIN macro instruction 128,
FREEPOOL macro instruction 73
functional characters 193-196

GATE macro instructions
(see communication with SYSIN/SYSOUT)
GATRD macro instruction 179
GATWR macro instruction 182
GDV macro instruction 211
generation name 15
absolute 15
relative 15
generation of literals 248-252
GET macro instruction
for QSAM 84
for VISAM 46
for VSAM 42
GETBUF macro instruction 70
GETMAIN macro instruction 126
GETPOOL macro instruction 72
GTWAR macro instruction 183
GTWSR macro instruction 184

HOLD (DDEF operand) 240

ICB
(see interrupt control block)
IMPLICIT adcon group 133,136
implicit linkage 228
implied address, definition of 12
IMSK (DCB operand) 26,32,234
inhibiting interrupts
(see SAI macro instruction)
INOUT (OPEN option) 38
INPUT (OPEN option) 38
input/output request facilities
(see IOREQ)
integer, definition of 13
inter-task communication
(see VSEND and VSENDR macro instructions)
interlock, sharing 258,259
INTERNAL adcon group 133
interrupt control block
for SAEC macro instruction 160
for SEEC macro instruction 156
for SIEC macro instruction 165
for SPEC macro instruction i 154
for SSEC macro instruction 158
for STEC macro instruction 163
interrupt handling 149,253-255
INTINQ macro instruction 167
RAE 167
SAEC 158
SAI 166
SEEC 155
SIEC 164
SIR 150
SPEC 152
SSEC 157
STEC 161
AETD 170
CLATT 170
USATT 169
DIR 165
interval timer 200,203
INTINQ macro instruction 167
INVOKE macro instruction 148
IOREQ facility 95
CHECK macro instruction 98
IOREQ 95
VCCW 98,99
ITI macro instruction 172

JOBLIB 241
job library
(see JOBLIB)

KEYLEN (DCB operand) 233
keyword 8
keyword operand 8
keyword creation
(see BPKD macro instruction)

label, volume
(see LABEL)
LABEL (DDEF operand) 241
LEAVE (CLOSE operand) 120
LIBESRCH macro instruction 148
linkage
conventions 225
(see also CALL, SAVE, and RETURN macro instructions)
explicit 228
implicit 227

linking and loading 133
ADCOND macro instruction 137
ADCON 134
ARM 138
CALL 139,142
LOAD 142
DELETE 143
SAVE 144
RETURN 146
list form macro instruction, definition of 18
literals, generation of 248,252
load adcon group 136
LOAD macro instruction 142
loading a module 142
locate mode
(see GET and PUT macro instructions)
log, system 199
logical record length
(see LRECL)
LRECL (DCB operand) 26,28,235
LSCHP macro instruction 132

MACRF (DCB operand) 26,31,235
macro expansion, definition of 7
macro instruction language 7
macro instruction generation of literals 248,252
macro library 257
magnetic tape positioning 120,121
MCAST macro instruction 193,196
manipulating entire data sets 101
copying data sets 101
bulk output 104
member interlock 258,259
MEND statement 258
mixed operand 8
MODE (DCB operand for card reader or card punch) 26,31,234
move mode
(see GET and PUT macro instructions)
MSGWR macro instruction 191

name, definition of 10,11
name field 7
for DCB 232
for DDEF 237
NCP (DCB operand) 26,32,235
notational symbols 8
NOTE macro instruction 80
use with POINT 80
O-type macro instructions 20
OBEY macro instructions 177
OPEN macro instruction 37,38-40
parameter list 260
OPEN options 38
operand
definition of 7
field 7
forms
keyword 8
mixed 8
oplist 16
positional 7
sublist 8
operator communication with
(see WTO and WTOR macro instructions)
oplist operands 16
OPTION (DDEF operand) 241
OPTCD (DCB operand) 26,32,35
OUTIN (OPEN option) 38
OUTPUT (OPEN option) 38

PAD (DCB operand) 26,29,235
page interlock 258,259
parameter area 227
parameter list 7,227
parameter, definition of 7
parameter register 7
parentheses, use of 9
partially qualified name 16
partitioned data set name 14
partitioned organization directory 55-57
PAUSE macro instruction 173
PCSOUT (dump routine data set) 237
PCSVC macro instruction 172
POD
(see partitioned organization directory)
POINT macro instruction 78
use with NOTE macro instruction 78
positional operand 7
PR macro instruction 104,107
printing a data set
(see PR and WT macro instructions)
private volumes
(see PRIVATE)
PRIVATE (DDEF operand) 240
PRMPT macro instruction 188
profile character and switch table 179
program management 126
providing symbolic names for DCB fields
(see DCBD macro instruction)
PRTOV macro instruction
for BSAM 81
for QSAM 91
PRTSP (DCB operand for printer spacing) 26,30,233
PTI macro instruction 172
PU macro instruction 107-109
punching a data set
(see PU macro instruction)
PUT macro instruction
for QSAM 85
for VISAM 47
for VSAM 43
PUTX macro instruction
for QSAM 86
for VSAM 44

queued sequential access method 83
QSAM macro instructions 83
GET 84
PUT 85
PUTX 86
RELSE 89
TRUNC 89
CNTRL 90
PRTOV 91
SETL 92

R-type macro instruction 17
RAE macro instruction 167
RDBACK (OPEN option) 38
READ macro instruction
  for BSAM 62
  for VISAM 48
reading commands from SYSIN
  (see CLIC and CLIP macro instructions)
RECFM (DCB operand) 26,27,235
record format (see RECFM)
REDTIM macro instruction 207
register notation 11
register type macro instruction 17
register use, for linkage 226
REL macro instruction 124
relative
  generation name 15
  key position (see RKP)
releasing
  a data set 124
  exclusive control of a record
  (see RELEX macro instruction)
  virtual storage 73
relexp, definition of 10,11
relocatable expression, definition
 of 11,12
RELSE macro instruction 89
removing data sets 123
  ERASE 123
  REL 124
removing a job library 125
replacing a record
  (see PUTX macro instruction)
REREAD (CLOSE operand) 120
restoring registers
  (see RESUME and RETURN macro
   instructions)
RESUME macro instruction 148
retention period
  (see RETPD)
RETPD (DDEF operand) 238,241
retrieving DDEF commands
  (see CDD macro instruction)
RETURN macro instruction 146
RKP (DCB operand) 26,29,231,236
RSPRV macro instruction 132
RTRN macro instruction 178

S-type, E-form macro instruction 19
S-type, L-form macro instruction 18
S-type macro instruction 18
SAEC macro instruction 158
SAI macro instruction 166
save area 227
  (see also SAVE macro instruction)
SAVE macro instruction 144
saving register contents
  (see SAVE macro instruction)
SEEC macro instruction 155
SETL macro instruction
  for QSAM 92
  for VISAM 51
  for VSAM 44
sharing data sets 258
SIC (operand for GATE macro
 instructions) 180,182,183,184
SIEC macro instruction 164
SIR macro instruction 150
SPACE (DDEF operand) 239
SPEC macro instruction 152
special symbol, definition of 16
specify
  interrupt routine (see SIR macro
   instruction)
  privilege class 131
specsym, definition of 10,11
SSEC macro instruction 157
STACK (DCB operand for stacker
 selection) 233
STEC macro instruction 161
STIMER macro instruction 200
storage control section
  (see CSTORE macro instruction)
Storage-type macro instruction 18
STORE macro instruction 148
STOW macro instruction 57
sublist, operand 8
symbol, definition of 12
  operand form 12
  value mnemonic 10,11
symbolic library 257
SYNAD (DCB operand) 26,29,236
SYNAD routine 218
  entry during BSAM or QSAM
   operations 218
  entry during VISAM operations 221
SYSEARCH routine 257,259
SYSIN macro instruction 185
SYSIN parameter 195
SYSINX parameter 186
SYSINDEX routine 257-259
SYSOUT 179-188
system macro instructions 7
SYSXBLD ROUTINE 256

terminal I/O 179-188,173
text, definition of 14
  operand form 14
  value mnemonic 10,11
timing maintenance 200
  EBCDTIME 204
  STIMER 200
  TTIMER 203
transfer to command mode 173
  PAUSE 173
  COMMAND 174
  EXIT 175
  ABEND 176
  OBEY 177
translation codes (see SIC)
TRK (DDEF operand) 240
TRTCH (DCB operand) 26,30,233
TRUNC macro instruction 89
TSKABEND data set 176
TTIMER macro instruction 203
TWAIT macro instruction 213

uncataloging data sets
  (see DEL macro instruction)

underscore, use of 9
UNIT (DDEF operand) 239
unit affinity
(see AFF)
unloading a module
(see DELETE macro instruction)
UPDAT (OPEN option) 38
upper-case letters, use of 9
USATT macro instruction 169

value, definition of 10,11
value mnemonics 9
VCCW macro instruction 98
vertical stroke 8
virtual channel command word
(see VCCW macro instruction)
virtual indexed sequential access method
(see VISAM)
virtual partitioned access method
(see VPAM)
virtual sequential access method
(see VSAM)
virtual storage management 126
GETMAIN 126-128
FREEMAIN 128-129
CSTORE 130,131
VISAM 46
GET 46
PUT 47
READ 48
WRITE 49
SETL 51
ESETL 52
DELREC 52
RELEX 53
volseqno (DDEF operand) 240
volserno (DDEF operand) 240
VOLUME (DDEF operand) 240
volume
label (see LABEL)
serial number (see volserno)
VPAM 55
FIND 55
STOW 57
VSAM 42
GET 42
PUT 43
PUTX 44
SETL 44
VSEND macro instruction 213
VSENDR macro instruction 213

WRITE macro instruction
for BSAM 64
for VISAM 49
writing tape
(see WT macro instruction)
WT macro instruction 109-112
WTL macro instruction 199
WTO macro instruction 197
WTOR macro instruction 198

XTRCT macro instruction 213
XTRSYS macro instruction 213
XTRXTS macro instruction 213

READER'S COMMENT FORM

IBM System/360 Time Sharing System
Assembler User Macro Instructions

C28-2004-2

- Your comments, accompanied by answers to the following questions, help us produce better publications for your use. If your answer to a question is "No" or requires qualification, please explain in the space provided below. Comments and suggestions become the property of IBM.

	Yes	No
• Does this publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

- What is your occupation? _____
- How do you use this publication?

As an introduction to the subject?	☐	As an instructor in a class?	☐
For advanced knowledge of the subject?	☐	As a student in a class?	☐
For information about operating procedures?	☐	As a reference manual?	☐

Other _____

- Please give specific page and line references with your comments when appropriate. If you wish a reply, be sure to include your name and address.

COMMENTS:

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS PLEASE . . .

This publication is one of a series which serves as reference for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 34
YORKTOWN HTS., NY

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM Corporation
PO Box 344
2651 Strang Boulevard
Yorktown Heights, N.Y. 10598

ATTN: Time Sharing System/360
Programming Publications Dept. 561

Fold Fold

Printed in U.S.A. C28-2004-2

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

C28-2004-2

Printed in U.S.A. C28-2004-2

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]