

**HONEYWELL**

DPS 6

GCOS 6 MOD 400

APPLICATION

DEVELOPER'S

GUIDE

**SOFTWARE**





**DPS 6  
GCOS 6 MOD 400 APPLICATION  
DEVELOPER'S GUIDE  
ADDENDUM B**

**SUBJECT**

Additions and Changes to the Manual

**SPECIAL INSTRUCTIONS**

This is the second addendum to CZ15-02, dated March 1986. Insert the attached pages into the manual according to the collating instructions on the back of this sheet. Change bars in the margin indicate new or changed information; asterisks indicate deletions.

**Note:**

Insert this cover sheet behind the front cover to indicate the updating of the document with Addendum B.

**SOFTWARE SUPPORTED**

This manual supports Release 4.0 through Update 02 of the MOD 400 Executive. For later versions of the executive that this manual may support, see the *ONE PLUS Guide to Software Documentation* (Order No. HE01).

**ORDER NUMBER**

CZ15-02B

March 1987

46664  
0287  
Printed in U.S.A.

**Honeywell**

## COLLATING INSTRUCTIONS

To update this manual, remove old pages and insert new pages as follows:

### Remove

v, vi  
2-5 through 2-10  
2-11, blank  
6-11 through 6-14  
6-33, 6-34  
6-81, 6-82  
9-5, 9-6  
h-1, blank

### Insert

v, vi  
2-5 through 2-10  
—  
6-11 through 6-14  
6-33, 6-34  
6-81, 6-82  
9-5, 9-6  
h-1, blank

**USER COMMENTS FORMS** are included at the back of this manual. These forms are to be used to record any corrections, changes, or additions that will make this manual more useful.

Honeywell disclaims the implied warranties of merchantability and fitness for a particular purpose and makes no express warranties except as may be stated in its written agreement with and for its customer.

In no event is Honeywell liable to anyone for any indirect, special or consequential damages. The information and specifications in this document are subject to change without notice. Consult your Honeywell Marketing Representative for product or service availability.

**DPS 6  
GCOS 6 MOD 400 APPLICATION  
DEVELOPER'S GUIDE  
ADDENDUM A**

**SUBJECT**

Additions and Changes to the Manual

**SPECIAL INSTRUCTIONS**

This is the first addendum to CZ15-02, dated March 1986. Insert the attached pages into the manual according to the collating instructions on the back of this sheet. Change bars in the margin indicate new or changed information; asterisks indicate deletions; except in Sections 6, 9, and 10 which have been extensively revised and change indicators have been omitted.

**Note:**

Insert this cover sheet behind the front cover to indicate the updating of the document with Addendum A.

**SOFTWARE SUPPORTED**

This manual supports Release 4.0 of the MOD 400 Executive.

**ORDER NUMBER**

CZ15-02A

September 1986

45989  
0986  
Printed in U.S.A.

**Honeywell**

## COLLATING INSTRUCTIONS

To update this manual, remove old pages and insert new pages as follows:

| <b>Remove</b>      | <b>Insert</b>            |
|--------------------|--------------------------|
| v through viii     | v through xviii          |
| ix, blank          | 2-3 through 2-6          |
| xi through xxiv    |                          |
| xxv, blank         |                          |
| 2-3 through 2-6    |                          |
| 3-11, 3-12         | 3-11, 3-12               |
| 3-15, 3-16         | 3-15, 3-16 3-12.1, blank |
| 6-1 through 6-104  | 6-1 through 6-104        |
| 6-105, blank       | 6-105, blank             |
| 7-1, 7-2           | 7-1, 7-2                 |
| 7-9 through 7-12   | 7-9 through 7-12         |
| 7-19, 7-20         | 7-19, 7-20               |
| 7-25 through 7-28  | 7-25 through 7-26        |
| 8-1 through 8-14   | 8-1 through 8-14         |
| 8-23, 8-24         | 8-23, 8-24               |
| 8-33, 8-34         | 8-33, 8-34               |
| 8-39, 8-40         | 8-39, 8-40               |
| 8-45, 8-46         | 8-45, 8-46               |
| 8-57 through 8-62  | 8-56.1, 8-56.2           |
|                    | 8-57 through 8-62        |
| 9-1 through 9-56   | 9-1 through 9-34         |
| 9-57, blank        | 10-1 through 10-48       |
| 10-1 through 10-40 |                          |
| 12-1 through 12-28 | -                        |
|                    | h-1                      |
| i-1 through i-14   | i-1 through 1-12         |
| i-15, blank        | i-13, blank              |

**USER COMMENTS FORMS** are included at the back of this manual. These forms are to be used to record any corrections, changes, or additions that will make this manual more useful.

Honeywell disclaims the implied warranties of merchantability and fitness for a particular purpose and makes no express warranties except as may be stated in its written agreement with and for its customer.

In no event is Honeywell liable to anyone for any indirect, special or consequential damages. The information and specifications in this document are subject to change without notice. Consult your Honeywell Marketing Representative for product or service availability.

**DPS 6  
GCOS 6 MOD 400  
APPLICATION DEVELOPER'S GUIDE**

**SUBJECT**

MOD 400 System Usage for Application Programmers

**SPECIAL INSTRUCTIONS**

This manual supersedes the *DPS 6 GCOS 6 MOD 400 Application Developer's Guide*, dated July 1984. Since the manual has been extensively revised and reorganized, change bars are not used.

**SOFTWARE SUPPORTED**

This manual supports Release 4.0 of the MOD 400 Executive.

**ORDER NUMBER**

CZ15-02

March 1986

**Honeywell**

## ***PREFACE***

This manual is written for the applications programmer. Its purpose is to provide the information needed to write and run application programs using the GCOS 6 MOD 400 operating system.

The reader should have a basic knowledge of application development and processing as well as some programming experience in COBOL, BASIC, or FORTRAN.

The major topics presented in this manual are

- Terminal startup and user access procedures
- File management
- Screen Editor conventions, directives, and user procedures
- Line Editor conventions, directives, and user procedures
- Compile, link, and execute procedures
- Program debug utility user procedures
- Patch utility user procedures
- Memory dump interpretation procedures.

USER COMMENTS FORMS are included at the back of this manual. These forms are to be used to record any corrections, changes, or additions that will make this manual more useful.

Honeywell disclaims the implied warranties of merchantability and fitness for a particular purpose and makes no express warranties except as may be stated in its written agreement with and for its customer.

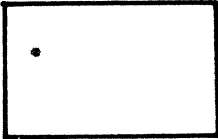
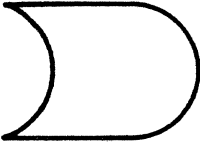


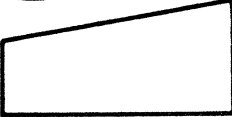

In no event is Honeywell liable to anyone for any indirect, special or consequential damages. The information and specifications in this document are subject to change without notice. Consult your Honeywell Marketing Representative for product or service availability.

After reading this manual, the applications programmer should be familiar with the MOD 400 system services and be able to write, debug, and run application programs.

The notation conventions used in this manual follow. The first set of conventions applies to directive syntax as a whole; the second set applies to flow chart symbols; the third set applies to heading hierarchies; and the fourth set applies to user keyins.

| <u>Syntax Convention</u> | <u>Meaning</u>                                    |
|--------------------------|---|
| UPPERCASE CHARACTERS     | Reserved keyword or symbol. Enter as shown.       |
| lowercase characters     | Variable field. Replace by a user-supplied value. |
| Brackets                 | Include none or one of the enclosed options.      |
| Braces                   | Include one of the enclosed options.              |

Flowchart Symbols

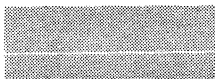
| <u>Flowchart Symbols</u>  | <u>Meaning</u>   |
|---|--|
|   | Process--represents performance of a computer operation.   |
|  | Online storage--represents information stored on diskette, cartridge disk, or storage module.        |
|  | Printed card--represents card input.   |
|  | Document--represents printed output.   |
|  | Manual input--terminal input.  |
|  | Mandatory--indicates that designated flow of information, type of processing, or output is required. |

## Heading Hierarchy

The following conventions are used to indicate the relative levels of topic headings used in this manual.

|                   |   |
|-------------------|---|
| Level 1 (highest) | <u>ALL CAPITALS, UNDERLINED</u>                 |
| Level 2           | <u>Initial Capitals, Underlined</u>             |
| Level 3           | ALL CAPITALS, NOT UNDERLINED                    |
| Level 4           | Initial Capitals, Not Underlined                |
| Level 5           | <u>Initial Capitals, Underscored (indented)</u> |

## User Keyins



Indicates user input to the system.



# CONTENTS

|   | Page |
|---|------|
| SECTION 1 INTRODUCTION.....                           | 1-1  |
| System Facilities.....                                | 1-1  |
| Application Development Components.....               | 1-2  |
| SECTION 2 SYSTEM ACCESS.....                          | 2-1  |
| User Access Procedures.....                           | 2-1  |
| Connecting the Terminal to the Central Processor..... | 2-1  |
| Direct-Connect Terminal.....                          | 2-2  |
| Dialup Terminal.....                                  | 2-2  |
| Connecting a User to the Executive.....               | 2-2  |
| Login Terminal.....                                   | 2-2  |
| Manual Login Terminal.....                            | 2-3  |
| Banner Login.....                                     | 2-3  |
| Forms Login.....                                      | 2-4  |
| Direct Login Terminal.....                            | 2-6  |
| Non-Login Terminal.....                               | 2-6  |
| Procedures and Conventions after Access.....          | 2-8  |
| Sending Messages to the Operator.....                 | 2-8  |
| Interrupting A Task.....                              | 2-8  |
| SECTION 3 FILE CONVENTIONS.....                       | 3-1  |
| Overview.....   | 3-1  |
| Disk File Conventions.....                            | 3-2  |
| Directories.....                                      | 3-2  |
| Root Directory.....                                   | 3-3  |
| System Root Directory.....                            | 3-3  |
| User Root Directories.....                            | 3-3  |
| Intermediate Directories.....                         | 3-3  |
| Working Directory.....                                | 3-4  |
| Locations of Disk Directories and Files.....          | 3-5  |
| Naming Conventions.....                               | 3-5  |
| Uniqueness of Names.....                              | 3-6  |
| Pathname.....   | 3-6  |
| Symbols Used in Pathnames.....                        | 3-6  |
| Absolute and Relative Pathnames.....                  | 3-8  |

\*

# CONTENTS

|   | Page   |
|---|--------|
| Magnetic Tape File Conventions.....             | 3-11   |
| 9-Track Magnetic Tape File Organization.....    | 3-11   |
| Magnetic Tape File and Volume Names.....        | 3-11   |
| Magnetic Tape Device Pathname Construction..... | 3-12   |
| Unlabeled Tape Pathnames.....                   | 3-12   |
| Labeled Tape Pathnames.....                     | 3-12   |
| Automatic Tape Volume Recognition.....          | 3-12.1 |
| Unit-Record Device File Conventions.....        | 3-12.1 |
| Working with Files.....                         | 3-13   |
| Command Processor.....                          | 3-13   |
| Standard I/O Files.....                         | 3-13   |
| Command Level.....                              | 3-14   |
| Controlling Your Operating Environment.....     | 3-14   |
| Volume Control.....                             | 3-15   |
| Creating Volumes.....                           | 3-15   |
| Renaming Disk Volumes.....                      | 3-16   |
| Directory Control.....                          | 3-16   |
| Changing Your Working Directory.....            | 3-16   |
| Creating Directories.....                       | 3-17   |
| Renaming Directories.....                       | 3-19   |
| Deleting Directories.....                       | 3-19   |
| File Control.....                               | 3-19   |
| Creating Files.....                             | 3-19   |
| Renaming Files.....                             | 3-21   |
| Deleting Files.....                             | 3-21   |
| Copying Files.....                              | 3-21   |
| Locating Files.....                             | 3-22   |
| Listing Files and Directories.....              | 3-23   |
| Interrupting Execution.....                     | 3-23   |
| Controlling Output.....                         | 3-24   |
| Directing Output to a File.....                 | 3-24   |
| Directing Output to a Printer.....              | 3-24   |
| Redirecting Output to Your Terminal.....        | 3-25   |
| Printing Control.....                           | 3-25   |
| Printing Files at Your Terminal.....            | 3-25   |
| Deferred Printing.....                          | 3-25   |
| Program Execution.....                          | 3-26   |
| Reserving Files or Devices.....                 | 3-28   |
| Communicating With Other Users.....             | 3-28   |
| <br>  |        |
| SECTION 4  SCREEN EDITOR.....                   | 4-1    |
| <br>  |        |
| Overview.....                                   | 4-1    |
| Screen Editor Processing.....                   | 4-2    |
| Screen Editor Suffix Conventions..              | 4-3    |
| Loading the Screen Editor.....                  | 4-3    |
| Description of the Screen.....                  | 4-5    |

# CONTENTS

|   | Page |
|---|------|
| Status Region.....                              | 4-6  |
| Text Region.....                                | 4-6  |
| Directive Region.....                           | 4-6  |
| Creating a Source Unit.....                     | 4-7  |
| Changing an Existing Source Unit.....           | 4-7  |
| Interrupting Screen Editor Processing.....      | 4-8  |
| Entering Screen Editor Directives.....          | 4-9  |
| Screen Editor Directive Format Conventions..... | 4-9  |
| Designating Lines.....                          | 4-10 |
| Block Description.....                          | 4-10 |
| Special Characters.....                         | 4-13 |
| Summary of Screen Editor Directives.....        | 4-14 |
| Screen Editor Directives.....                   | 4-15 |
| BOTTOM LINE (BOTTOM LINE OR BL).....            | 4-16 |
| CHANGE (CHANGE OR C).....                       | 4-17 |
| CHANGE_ALL (CHANGE ALL OR CA).....              | 4-19 |
| CHANGE_BLOCK (CHANGE_BLOCK OR CB).....          | 4-21 |
| DISPLAY.....                                    | 4-23 |
| LANGUAGE_TYPE (LANGUAGE_TYPE OR LT).....        | 4-24 |
| LEFT_MARGIN (LEFT_MARGIN OR LM).....            | 4-25 |
| LOWER_CASE (LOWER_CASE OR LC).....              | 4-26 |
| QUIT (QUIT OR Q).....                           | 4-27 |
| READ (READ OR R).....                           | 4-28 |
| RIGHT_MARGIN (RIGHT_MARGIN OR RM).....          | 4-29 |
| SCROLL_CHANGE (SCROLL_CHANGE OR SC).....        | 4-30 |
| SEARCH (SEARCH OR S).....                       | 4-31 |
| SEARCH_BACKWARD (SEARCH_BACKWARD OR SB).....    | 4-33 |
| SEARCH_FORWARD (SEARCH_FORWARD OR SF).....      | 4-35 |
| TOP_LINE (TOP_LINE OR TL).....                  | 4-37 |
| TRAILING_BLANKS (TRAILING_BLANKS OR TB).....    | 4-38 |
| UPPER_CASE (UPPER_CASE OR UC).....              | 4-39 |
| VERSION (VERSION OR V).....                     | 4-40 |
| WINDOW_WIDTH (WINDOW_WIDTH OR WW).....          | 4-41 |
| WRITE (WRITE OR W).....                         | 4-42 |
| WRITE_BLOCK (WRITE_BLOCK OR WB).....            | 4-44 |
| Function Keys.....                              | 4-46 |
| APPEND LINE.....                                | 4-49 |
| BACKWARD WORD.....                              | 4-50 |
| BLOCK.....                                      | 4-51 |
| COPY BLOCK.....                                 | 4-53 |
| DELETE BLOCK.....                               | 4-55 |
| ERASE BLOCK.....                                | 4-56 |
| FORWARD WORD.....                               | 4-57 |
| MOVE BLOCK.....                                 | 4-58 |
| WINDOW DOWN.....                                | 4-60 |
| WINDOW LEFT.....                                | 4-61 |
| WINDOW RIGHT.....                               | 4-62 |

# CONTENTS

|   | Page |
|---|------|
| WINDOW UP.....  | 4-63 |
| Labeled Keys.....   | 4-64 |
| BACKSPACE.....  | 4-67 |
| CARRIAGE RETURN.....  | 4-68 |
| CLEAR/RESET.....  | 4-69 |
| CTL CLR/TAB/SET.....  | 4-70 |
| CTRL TAB.....   | 4-71 |
| CURSOR DOWN (↓).....  | 4-72 |
| CURSOR LEFT (←).....  | 4-73 |
| CURSOR RIGHT (→).....   | 4-74 |
| CURSOR UP (↑).....  | 4-75 |
| DEL CHAR.....   | 4-76 |
| DEL LINE.....   | 4-77 |
| ERASE EOL.....  | 4-78 |
| HOME.....   | 4-79 |
| INS CHAR.....   | 4-80 |
| INS LINE.....   | 4-82 |
| LINE FEED.....  | 4-83 |
| TAB.....  | 4-84 |
| TAB CLR.....  | 4-85 |
| TAB SET.....  | 4-86 |
| <br>  |      |
| SECTION 5 LINE EDITOR.....  | 5-1  |
| <br>  |      |
| Overview.....   | 5-1  |
| Line Editor Suffix Conventions.....   | 5-3  |
| Line Editor Directive Format Conventions.....   | 5-3  |
| Methods of Specifying Addresses.....  | 5-5  |
| Designating a Line Number as an Address.....  | 5-6  |
| Designating the Position of a Line Relative to the<br>"Current" Line as an Address..... | 5-6  |
| Designating Contents of Line as an Address.....   | 5-7  |
| Compound Addresses.....   | 5-11 |
| Referencing a Series of Lines.....  | 5-12 |
| Loading the Line Editor.....  | 5-14 |
| Summary of Line Editor Directives and Escape Sequences....                              | 5-16 |
| Creating a Source Unit.....   | 5-21 |
| Changing an Existing Source Unit.....   | 5-22 |
| Input Mode Description and Directives.....  | 5-22 |
| APPEND (A).....   | 5-24 |
| CHANGE (C).....   | 5-27 |
| INSERT (I).....   | 5-30 |
| Edit Mode Description and Directives.....   | 5-33 |
| DELETE (D).....   | 5-35 |
| PRINT (P).....  | 5-37 |
| QUIT (Q OR !Q).....   | 5-40 |
| READ (R).....   | 5-41 |

# CONTENTS

|   | Page  |
|---|-------|
| SUBSTITUTE (S OR !S).....                             | 5-44  |
| WRITE (W).....  | 5-48  |
| Advanced Functions of the Line Editor.....            | 5-50  |
| General Advanced Line Editor Directives.....          | 5-50  |
| EXCLUDE (V).....                                      | 5-51  |
| EXECUTE (E).....                                      | 5-53  |
| GLOBAL (G).....                                       | 5-54  |
| LINE FEED (L OR !L).....                              | 5-56  |
| LOWERCASE (U).....                                    | 5-57  |
| NEW CURRENT LINE (N).....                             | 5-58  |
| PRINT LINE NUMBER (=!/P).....                         | 5-59  |
| PRINT WITH LINE NUMBER (!P).....                      | 5-61  |
| UPPERCASE (!U).....                                   | 5-63  |
| COMMENT (").....                                      | 5-64  |
| Auxiliary Buffer Directives and Escape Sequences..... | 5-65  |
| ACCEPT SINGLE LINE FROM A TERMINAL (!R).....          | 5-67  |
| BUFFER STATUS (X).....                                | 5-68  |
| CHANGE BUFFER (Bx).....                               | 5-70  |
| CHANGE ORIGIN OF TEXT DURING EDIT MODE (!B).....      | 5-71  |
| CHANGE ORIGIN OF TEXT DURING INPUT MODE (!B).....     | 5-74  |
| COPY (K).....   | 5-76  |
| COPY-APPEND (!K).....                                 | 5-78  |
| DESTROY (^B).....                                     | 5-80  |
| MOVE (M).....   | 5-81  |
| MOVE-APPEND (!M).....                                 | 5-83  |
| Line Editor Debugging Directives.....                 | 5-85  |
| HEXADECIMAL DUMP (ZDUMP).....                         | 5-86  |
| ZREGEXP.....  | 5-88  |
| ZTRACE.....   | 5-89  |
| Line Editor Programming Directives.....               | 5-92  |
| ADDRESS PREFIX (?).....                               | 5-93  |
| GO TO (>).....  | 5-95  |
| IF DATA (#).....                                      | 5-97  |
| IF EMPTY (^#).....                                    | 5-98  |
| IF LINE (adr#).....                                   | 5-99  |
| IF NOT LINE (adr ^#).....                             | 5-100 |
| IF RANGE (addr(s) #).....                             | 5-101 |
| IF NOT RANGE (adrs ^#).....                           | 5-102 |
| LABEL (:).....  | 5-103 |
| SEARCH (*).....                                       | 5-104 |
| SEARCH NOT (^*).....                                  | 5-105 |
| TYPE (T).....   | 5-106 |
| Programming Considerations.....                       | 5-107 |
| Line Editor Procedures.....                           | 5-107 |
| Initiating a Line Editor Session.....                 | 5-108 |
| Creating Work Files.....                              | 5-108 |
| Line Editor Modes.....                                | 5-109 |

# CONTENTS

|  | Page  |
|--|-------|
| Quitting the Line Editor.....  | 5-109 |
| Creating a File.....   | 5-110 |
| Addressing Techniques.....   | 5-111 |
| Addressing a Single Line.....  | 5-111 |
| Addressing Multiple Lines.....                                       | 5-112 |
| Printing Line Numbers.....   | 5-112 |
| Use of Period (.) for Current Line.....                              | 5-113 |
| Character String Addressing.....                                     | 5-113 |
| Selective Specification of Character Strings.....                    | 5-114 |
| Specifying Initial Character String.....                             | 5-114 |
| Specifying a Character String Ending a Line.....                     | 5-114 |
| Specifying a Single Character Substitution in<br>Search Strings..... | 5-115 |
| Use of Escape Characters.....  | 5-115 |
| Saving File Contents.....  | 5-116 |
| Reading File Contents.....   | 5-117 |
| Deleting Lines in Current Buffer.....                                | 5-118 |
| Deleting Multiple Lines.....   | 5-118 |
| Deleting All Lines in Current Buffer.....                            | 5-118 |
| Avoiding Post-Deletion Problems.....                                 | 5-119 |
| Adding and Deleting Lines.....                                       | 5-120 |
| Changing Line Contents.....  | 5-120 |
| Changing Character Strings Within a Line.....                        | 5-120 |
| Changing All Occurrences of a String.....                            | 5-121 |
| Substituting Initial and Concluding Strings.....                     | 5-121 |
| Deleting Character Strings.....                                      | 5-122 |
| Appending a New String to an Existing String.....                    | 5-123 |
| Adding Lines to the Current Buffer.....                              | 5-123 |
| Inserting Lines.....   | 5-123 |
| Appending Lines.....   | 5-124 |
| Global Directives.....   | 5-124 |
| Global Delete.....   | 5-124 |
| Global Print.....  | 5-125 |
| Current and Auxiliary Buffers.....                                   | 5-125 |
| Repeating Lines in a File.....                                       | 5-126 |
| Moving Lines in a File.....  | 5-127 |
| Using Existing Files.....  | 5-128 |
| Buffer Status.....   | 5-129 |
| Saving Modified Buffer Contents.....                                 | 5-130 |
| Using System Commands in the Editor.....                             | 5-130 |
| Writing to Line Printer.....   | 5-130 |
| Date and Time.....   | 5-131 |
| Important Considerations.....  | 5-132 |
| SECTION 6 LINKER.....  | 6-1   |
| Linker Functions.....  | 6-1   |

# CONTENTS

|  | Page |
|--|------|
| Linker Directive Categories.....                           | 6-3  |
| Specifying Object Unit(s) to be Linked.....                | 6-3  |
| Specifying Location(s) of Object Unit(s) to be Linked...   | 6-3  |
| Creating a Root and Optional Overlay(s).....               | 6-4  |
| Producing Link Map(s).....                                 | 6-5  |
| Defining External Symbols.....                             | 6-6  |
| Protecting or Purging Symbol(s).....                       | 6-6  |
| Reloading After System Failure.....                        | 6-6  |
| Controlling the Directive File.....                        | 6-6  |
| Terminating the Linker.....                                | 6-7  |
| Loading the Linker.....                                    | 6-7  |
| Entering Linker Directives.....                            | 6-10 |
| Setting Access in the Linker's SEG and FSEG Directives.... | 6-11 |
| Setting Access for BMMU and EMMU Segments.....             | 6-11 |
| Setting Access for VMMU Segments.....                      | 6-11 |
| Linker Directives.....                                     | 6-13 |
| BASE (or BE).....  | 6-14 |
| CALL CANCEL (CC).....                                      | 6-21 |
| COMMON (or COMM).....                                      | 6-22 |
| CPROT (or CT).....   | 6-23 |
| CPURGE (or CE).....  | 6-24 |
| EDEF (or EF).....  | 6-25 |
| FLOATB6 (or F6).....                                       | 6-28 |
| FLOVLY (or FY).....  | 6-29 |
| FSEG (or FG).....  | 6-31 |
| GSHARE (or GE).....  | 6-33 |
| IN.....  | 6-34 |
| INCLUDE (or IE).....                                       | 6-36 |
| INIT2 (or I2).....   | 6-37 |
| IST (or IT).....   | 6-38 |
| LDEF (or LF).....  | 6-39 |
| LIB (or LIB1).....   | 6-43 |
| LIB 2, 3, 4.....   | 6-45 |
| LINK (or LK).....  | 6-46 |
| LINKN (or LN).....   | 6-47 |
| LINKnn.....  | 6-50 |
| LINKO (or LO).....   | 6-51 |
| LSR.....   | 6-52 |
| MAP, MAPD and MAPU (or MP, MD, and MU).....                | 6-53 |
| NOTCMD (or ND).....  | 6-64 |
| ONECPU (or OU).....  | 6-65 |
| OVERLAYTABLE (or OE or OT).....                            | 6-66 |
| OVLY (or OY).....  | 6-67 |
| PAGEPOOL (or PL).....                                      | 6-69 |
| PROTECT (PROT or PT).....                                  | 6-70 |
| PSU.....   | 6-72 |
| PURGE (or PE).....   | 6-73 |

# CONTENTS

|   | Page    |
|---|---------|
| QUIT (or QT or Q).....                                | 6-75    |
| REPORT (or RT).....                                   | 6-76    |
| RERUN RELOCATABLE (RR).....                           | 6-77    |
| RETURN (or RN).....                                   | 6-78    |
| SEG (or SG).....                                      | 6-79    |
| SHARE (or SE).....                                    | 6-82    |
| STACK (or SK).....                                    | 6-83    |
| START (or ST).....                                    | 6-84    |
| SWAPPOOL (or SL).....                                 | 6-85    |
| SYS (or SS).....                                      | 6-86    |
| UNPROTECT (or UNPROT or UT).....                      | 6-87    |
| USERPOOL (or UL).....                                 | 6-89    |
| VAL (or VL).....                                      | 6-90    |
| VDEF (or VF).....                                     | 6-91    |
| VPURGE (or VE).....                                   | 6-92    |
| Linker Procedures.....                                | 6-93    |
| Using Overlays.....                                   | 6-93    |
| Interrupting Linker Execution.....                    | 6-93    |
| Sample Link Sessions.....                             | 6-94    |
| Example 1: Linking With a Minimum of Directives.....  | 6-94    |
| Example 2: Specifying an Input Device.....            | 6-99    |
| Example 3: Linking More Than One Object Unit.....     | 6-99    |
| Example 4: Linking With Two Overlays.....             | 6-101   |
| <br>SECTION 7 MULTIUSER DEBUGGER (SYMBOLIC MODE)..... | <br>7-1 |
| Overview.....   | 7-1     |
| Capabilities.....                                     | 7-2     |
| Invoking the Debugger (Symbolic Mode).....            | 7-2     |
| Debugger and Break Key Functionality.....             | 7-6     |
| Planning Considerations.....                          | 7-7     |
| Controlling Execution of the User's Program.....      | 7-7     |
| Setting Breakpoints.....                              | 7-7     |
| Monitoring the Value of Variables.....                | 7-7     |
| Controlling Output.....                               | 7-7     |
| Maintaining a Trace History.....                      | 7-7     |
| Altering Values.....                                  | 7-7     |
| Debugger Directives.....                              | 7-8     |
| ACTIVATE (OR AC).....                                 | 7-9     |
| AT.....   | 7-10    |
| CHANGE (OR CH).....                                   | 7-12    |
| CLEAR (OR C).....                                     | 7-13    |
| DUMP (OR DP).....                                     | 7-14    |
| GO.....   | 7-15    |
| IF.....   | 7-16    |
| LIST (OR L).....                                      | 7-18    |
| MODE.....   | 7-19    |



# CONTENTS

|  | Page |
|--|------|
| PAUSE (OR P).....  | 7-20 |
| QUIT (QT).....   | 7-21 |
| SET.....   | 7-22 |
| SLEEP (SP).....  | 7-23 |
| TRACE (OR TR).....                                       | 7-24 |
| Multiuser Debugger (SYMBOLIC MODE) Procedures.....       | 7-26 |
| Compiling a Program For Use With the Debugger.....       | 7-26 |
| Sample Compilation Dialogs.....                          | 7-26 |
| Linking an Object Unit With the Debugger.....            | 7-27 |
| Sample Linker Dialogs.....                               | 7-27 |
| Invoking the Debugger.....                               | 7-28 |
| Sample Initialization Dialog.....                        | 7-29 |
| Debugging Multiple Bound Units.....                      | 7-30 |
| Executing Your Program With the Debugger.....            | 7-30 |
| Sample Executing Dialog.....                             | 7-30 |
| <br>   |      |
| SECTION 8 MULTIUSER DEBUGGER (NUMERIC MODE).....         | 8-1  |
| Overview.....  | 8-2  |
| Capabilities.....  | 8-2  |
| Invoking the Debugger (Numeric Mode).....                | 8-2  |
| Debugger File Requirements.....                          | 8-3  |
| Debugger Memory Requirements.....                        | 8-3  |
| Debugger Operation.....                                  | 8-3  |
| Entering Directives.....                                 | 8-4  |
| Debugger and Break Key Functionality.....                | 8-10 |
| Planning Considerations.....                             | 8-10 |
| Setting True Breakpoints and Bound Unit Breakpoints..... | 8-10 |
| Setting Global Breakpoints.....                          | 8-11 |
| Setting Quick Breakpoints.....                           | 8-11 |
| Preliminary Steps for Using Quick Breakpoints.....       | 8-11 |
| Guidelines for Setting Breakpoints.....                  | 8-12 |
| Controlling Output.....                                  | 8-12 |
| Determining the Active Level.....                        | 8-12 |
| Maintaining a Trace History.....                         | 8-13 |
| Debugger Directives.....                                 | 8-13 |
| ALL REGISTERS (AR).....                                  | 8-14 |
| ASSIGN (AS).....   | 8-15 |
| CHANGE MEMORY (CH).....                                  | 8-16 |
| CLEAR ABNORMAL TRAP BIT (CT).....                        | 8-17 |
| CLEAR ALL BOUND UNIT BREAKPOINTS (CB*).....              | 8-18 |
| CLEAR ALL QUICK BREAKPOINTS (CQ*).....                   | 8-19 |
| CLEAR ALL TRUE BREAKPOINTS (C*).....                     | 8-20 |
| CLEAR BOUND UNIT BREAKPOINT (CBn).....                   | 8-21 |
| CLEAR QUICK BREAKPOINT (CQn).....                        | 8-22 |
| CLEAR TRUE BREAKPOINT (Cn).....                          | 8-23 |
| CONDITIONAL EXECUTION (IF).....                          | 8-24 |

|  |         |
|--|---------|
| DEFINE (Dn).....                                     | 8-27    |
| DEFINE TRACE (DT).....                               | 8-28    |
| DISPLAY MEMORY (DH).....                             | 8-29    |
| DUMP MEMORY (DP).....                                | 8-30    |
| END TRACE (ET).....                                  | 8-31    |
| ESCAPE (E).....                                      | 8-32    |
| EXECUTE (En).....                                    | 8-33    |
| FILE IN (FI).....                                    | 8-34    |
| FILE OUT (FO).....                                   | 8-35    |
| GET QUICK MEMORY (MQ).....                           | 8-36    |
| GO.....  | 8-38    |
| LIST ALL BOUND UNIT BREAKPOINTS (LB*).....           | 8-39    |
| LIST ALL QUICK BREAKPOINTS (LQ*).....                | 8-40    |
| LIST ALL TRUE BREAKPOINTS (L*).....                  | 8-41    |
| LIST BOUND UNIT BREAKPOINT (LBn).....                | 8-42    |
| LIST QUICK BREAKPOINT (LQn).....                     | 8-43    |
| LIST TRUE BREAKPOINT (Ln).....                       | 8-44    |
| MODE.....  | 8-45    |
| PRINT (Pn).....                                      | 8-46    |
| PRINT ALL (P*).....                                  | 8-47    |
| PRINT HEADER LINE (Hn).....                          | 8-48    |
| PRINT HEXADECIMAL VALUE (VH).....                    | 8-49    |
| PRINT QUICK MEMORY POINTER (PQ).....                 | 8-50    |
| PRINT TRACE (PT).....                                | 8-51    |
| QUIT (QT).....                                       | 8-52    |
| RESET FILE (RF).....                                 | 8-53    |
| RETURN QUICK MEMORY (RQ).....                        | 8-54    |
| SET BOUND UNIT BREAKPOINT (SBn).....                 | 8-55    |
| SET EXPRESS BOUND UNIT BREAKPOINT (XBn).....         | 8-56.1  |
| SET QUICK BREAKPOINT (SQn).....                      | 8-58    |
| SET TRUE BREAKPOINT (Sn).....                        | 8-61    |
| SLEEP (SP).....                                      | 8-63    |
| SPECIFY FILE (SF).....                               | 8-64    |
| START J-MODE TRACE (ST).....                         | 8-67    |
| TURN ON ABNORMAL TRAP BIT (TB).....                  | 8-68    |
| TERMINATE THE TRAPPED TASK (TT).....                 | 8-69    |
| Multiuser Debugger (Numeric Mode) Procedures.....    | 8-70    |
| Sample Session 1.....                                | 8-70    |
| Sample Session 2.....                                | 8-79    |
| Sample Session 3.....                                | 8-86    |
| <br>SECTION 9 REQUESTING AND USING MEMORY DUMPS..... | <br>9-1 |
| Overview.....  | 9-1     |
| Using the Dump Utilities.....                        | 9-2     |
| Creating a Dump Volume.....                          | 9-3     |
| Setting Dumpfile Size.....                           | 9-4     |

# CONTENTS

|   | Page     |
|---|----------|
| Dumpfile Format.....  | 9-4      |
| Determining Available Disk Space.....   | 9-5      |
| Maximum Dumpfile Size.....  | 9-5      |
| Multiple-Volume Diskette.....   | 9-5      |
| Shared Dump and System Volumes.....   | 9-7      |
| Taking a Dump Using a Control Panel.....  | 9-8      |
| Taking a Dump Using the System Control Facility.....                            | 9-9      |
| Using DPEDIT.....   | 9-10     |
| Page Header.....  | 9-11     |
| Line Format.....  | 9-11     |
| Physical Dump.....  | 9-12     |
| Logical Dump.....   | 9-12     |
| System Summary.....   | 9-12     |
| Task Related Information.....   | 9-14     |
| Memory Pool Structures.....   | 9-15     |
| Task Group Structures.....  | 9-15     |
| Task Structures.....  | 9-16     |
| DPEDIT Command.....   | 9-17     |
| Operating Procedure for DPEDIT.....   | 9-19     |
| Interpreting and Using Memory Dumps.....  | 9-19     |
| Determining the State of Execution of Your Code at the<br>Time of the Dump..... | 9-21     |
| Halt at Level 2.....  | 9-21     |
| User Level Active at the Time of Dump.....                                      | 9-22     |
| No Level Active at the Time of Dump.....  | 9-22     |
| Locating a Trap Processed by the System Default Handler.....                    | 9-22     |
| Using XRAY.....   | 9-23     |
| Accessible Structures.....  | 9-23     |
| XRAY Command.....   | 9-24     |
| Operating Procedure for XRAY.....   | 9-27     |
| <br>SECTION 10 PATCH UTILITY.....   | <br>10-1 |
| Overview.....   | 10-1     |
| Operation.....  | 10-2     |
| Absentee Mode.....  | 10-2     |
| Interactive Mode.....   | 10-3     |
| Loading Patch.....  | 10-3     |
| Submitting Patch Directives.....  | 10-4     |
| Patching Techniques.....  | 10-7     |
| Naming the Patch.....   | 10-7     |
| Applying the Patch.....   | 10-7     |
| Patch Directives.....   | 10-7     |
| CLEAR SYSTEM BIT (CLSY).....  | 10-8     |
| COMMENT (*).....  | 10-9     |
| DATA PATCH (DP).....  | 10-10    |
| ELIMINATE PATCH (EP).....   | 10-14    |

# CONTENTS

Page

|  |       |
|--|-------|
| GO.....  | 10-16 |
| GROUP PATCH (GP).....                                | 10-17 |
| HEXADECIMAL PATCH (HP).....                          | 10-18 |
| INTERROGATE BOUND UNIT (WA).....                     | 10-22 |
| LDEF.....  | 10-23 |
| LIST GROUP PATCH NAMES (LG).....                     | 10-25 |
| LIST SPECIFIED GROUP PATCH (LG).....                 | 10-26 |
| LIST PATCHES (LP).....                               | 10-27 |
| LIST PATCHES NOW (LN).....                           | 10-29 |
| LIST PATCH NAMES (LS).....                           | 10-30 |
| LIST SPECIFIED PATCH (LS).....                       | 10-32 |
| LIST UPDATES (LU).....                               | 10-33 |
| QUIT (Q).....  | 10-34 |
| SET GLOBAL SHARE BIT OFF (GNSH).....                 | 10-35 |
| SET GLOBAL SHARE BIT ON (GSHR).....                  | 10-36 |
| SET SHARE BIT OFF (NS).....                          | 10-37 |
| SET SHARE BIT ON (SS).....                           | 10-38 |
| SET SYSTEM BIT ON (STSY).....                        | 10-39 |
| SYMBOLIC DATA PATCH (SD).....                        | 10-40 |
| SYMBOLIC PATCH (SP).....                             | 10-43 |
| VDEF.....  | 10-48 |
| <br>   |       |
| SECTION 11 MESSAGES.....                             | 11-1  |
| <br>   |       |
| Message Reporter.....                                | 11-1  |
| Message Libraries.....                               | 11-2  |
| System Message Library.....                          | 11-2  |
| Group Libraries.....                                 | 11-2  |
| Primary Libraries.....                               | 11-3  |
| Message Format.....                                  | 11-3  |
| Message Code.....                                    | 11-4  |
| Indicator Field.....                                 | 11-4  |
| Chain Pointer.....                                   | 11-4  |
| Message Text.....                                    | 11-5  |
| Parameter Designators.....                           | 11-5  |
| Parameterized Messages.....                          | 11-5  |
| Parameter Designator Format.....                     | 11-6  |
| Message Chaining.....                                | 11-10 |
| Standard Messages in the System Message Library..... | 11-12 |
| Message Library Utilities.....                       | 11-13 |
| ZXDSMG Utility.....                                  | 11-13 |
| ZXBTMG Utility.....                                  | 11-13 |
| Argument Definition.....                             | 11-14 |
| Updating the Message Library.....                    | 11-14 |
| Message Structure.....                               | 11-14 |
| Adding a Message to the Message Library.....         | 11-15 |
| National Language Support.....                       | 11-18 |

## CONTENTS

|                       | Page |
|-----------------------|------|
| MANUAL DIRECTORY..... | h-i  |
| INDEX.....            | i-1  |

## ILLUSTRATIONS

| Figure   | Page  |
|--|-------|
| 1-1 Application Development Component.....   | 1-3   |
| 2-1 Login Form.....  | 2-5   |
| 2-2 Login Arguments Form.....  | 2-6   |
| 2-3 Sample Directory Listing.....  | 2-7   |
| 3-1 Example of Disk File Directory Structure.....  | 3-2   |
| 3-2 Sample Directory Structure.....  | 3-4   |
| 3-3 Sample Pathnames.....  | 3-10  |
| 3-4 Location of Directories SHEPARD and COOK.....  | 3-18  |
| 3-5 Location of Subordinate File REPORTS.....  | 3-20  |
| 3-6 Location of Subordinate File WORDLIST.....   | 3-21  |
| 4-1 Sample Screen for Creating a File.....   | 4-5   |
| 4-2 Sample Screen for Modifying a File.....  | 4-5   |
| 4-3 Screen Editor Template for VIP780X General Purpose<br>Keyboard.....                          | 4-46  |
| 4-4 Screen Editor Template for VIP730X and HDS 2 General<br>Purpose and Data Entry Keyboard..... | 4-47  |
| 4-5 Screen Editor Template for VIP7300 and VIP7800 Word<br>Processing Keyboard.....              | 4-47  |
| 4-6 Screen Editor Template for microSystem 6/10<br>Keyboard.....                                 | 4-48  |
| 4-7 Screen Editor Template for VIP7200 Keyboard.....   | 4-48  |
| 4-8 Screen Editor Template for VIP7201 Keyboard.....   | 4-48  |
| 6-1 Relative Location of Memory in Memory Pool AA.....   | 6-20  |
| 6-2 Overlays in Memory Pool AA.....  | 6-20  |
| 6-3 Link Map Formats.....  | 6-56  |
| 6-4 Sample Link Map (DATIME.M).....  | 6-96  |
| 6-5 Structure of the Bound Unit SAMPLE.....  | 6-102 |
| 6-6 Source Listing of PROG.....  | 6-102 |
| 6-7 Source Listing of First Overlay Module PROG0.....  | 6-103 |
| 6-8 Source Listing of Second Overlay Module PROG1.....   | 6-103 |

## ILLUSTRATIONS

| Figure |  | Page |
|--------|--|------|
| 8-1    | Sample Program TEST.....                   | 8-71 |
| 8-2    | Debugging Session of TEST.....             | 8-73 |
| 8-3    | Bound Unit TSTNOW.....                     | 8-80 |
| 8-4    | Debugging Session of TSTNOW.....           | 8-82 |
| 8-5    | Contents of Quick Disk File TSTNOW.QK..... | 8-85 |
| 8-6    | Dump of Quick Memory.....                  | 8-87 |
| 8-7    | Debugging Session (Example 3).....         | 8-88 |
| 8-8    | Dump of Quick Disk File Sample .QK.....    | 8-91 |
| 11-1   | Message Library Record Structure.....      | 11-4 |

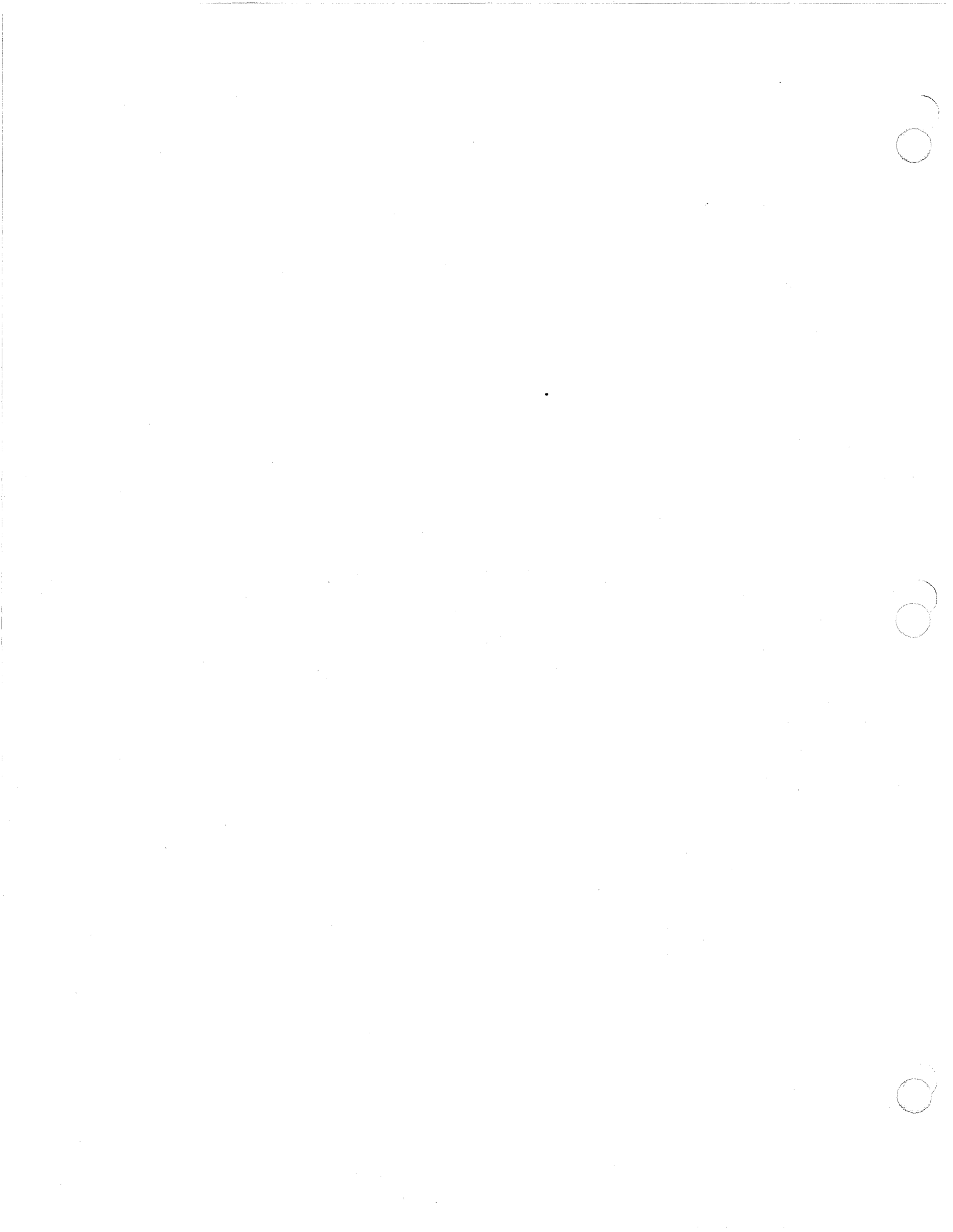
## TABLES

| Table |  | Page |
|-------|--|------|
| 4-1   | Summary of Screen Editor Directives.....                       | 4-14 |
| 4-2   | Correlation of SCORPEO's Labeled Keys.....                     | 4-66 |
| 5-1   | Summary of Line Editor Directives and Escape<br>Sequences..... | 5-16 |
| 7-1   | Summary of Symbolic Mode Directives.....                       | 7-3  |
| 7-2   | Terms Used in Symbolic Mode Directives.....                    | 7-4  |
| 7-3   | Symbolic Mode Special Symbols.....                             | 7-5  |
| 8-1   | Summary of Numeric Mode Directives.....                        | 8-5  |
| 8-2   | Symbols Used in Numeric Mode Directive Lines.....              | 8-6  |
| 9-1   | Significant Locations on Memory Dump.....                      | 9-20 |

REMOVE THIS PAGE AND PLACE TAB FOR

TAB 1

INTRODUCTION





# *Section 1*

## **INTRODUCTION**

The DPS 6 GCOS 6 MOD 400 Application Developer's Guide describes the GCOS 6 system facilities available to the application programmer and provides the procedures to write, debug, and run application programs.

### SYSTEM FACILITIES

The GCOS 6 MOD 400 Executive supports concurrent execution of one or more online job streams.

User-written online applications can be loaded and started at any time after system initialization. The number of applications in operation is determined by the amount of available memory. When one application is deleted or terminates, its memory is automatically released to another application.

The MOD 400 Executive allocates memory dynamically from pools and can relocate programs at load time. Once an application is loaded into memory, it is dispatched according to its assigned priority level. When multiple tasks share a priority level, they are serviced in a round-robin fashion. The Memory Management Unit (MMU) prevents user applications residing in different memory pools from interfering with each other or with the Executive.

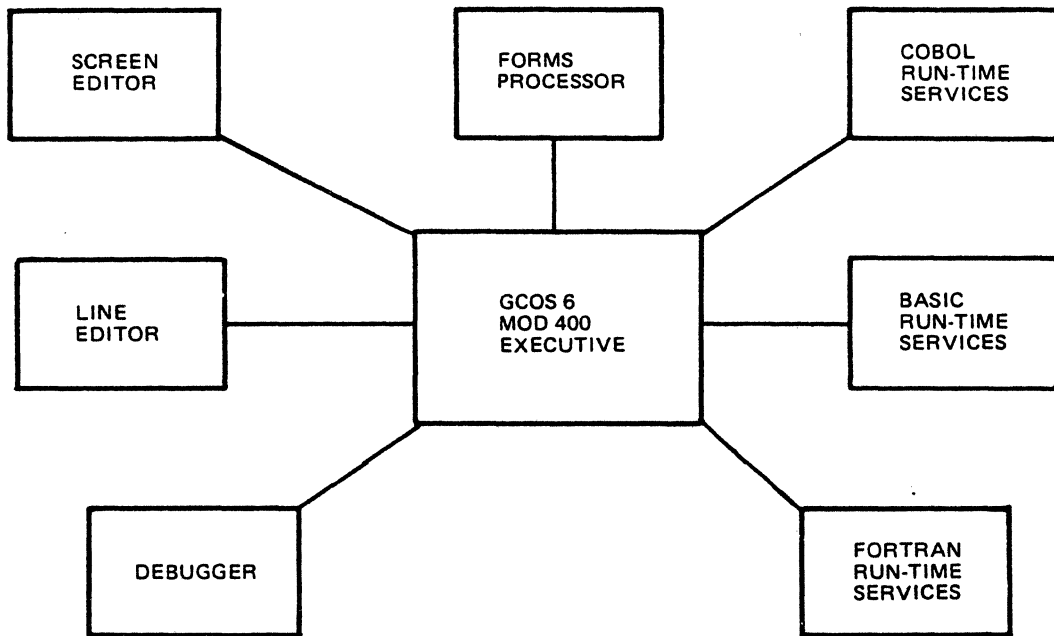
## APPLICATION DEVELOPMENT COMPONENTS

The following components support application development:

- Screen Editor--A full screen, interactive program development, text editing, and documentation preparation system that allows a user to enter an entire screen of data into a work file. The ability to manipulate full screens of data at once makes text editing faster and reduces I/O processing.
- Line Editor--An interactive program development, text editing, and documentation preparation system that works on data a line at a time.
- COBOL, BASIC, and FORTRAN Run-Time Services--A total system of language processors including compile, link, and execute modules that validate and process COBOL, BASIC, and FORTRAN programs.
- Forms Processor--A software component that permits a programmer to define terminal screen layouts as well as control characteristics of the data transmitted between the terminal and program variable storage.
- Debugger--A software diagnostic tool used to debug programs.

Figure 1-1 illustrates the application development components.

The Screen Editor, Line Editor, COBOL, BASIC, and FORTRAN run-time services, and the Multiuser Debugger are described in this manual. Forms processing is described in the Display Formatting and Control manual. The MOD 400 Executive is described in the MOD 400 System Concepts manual.



86-139

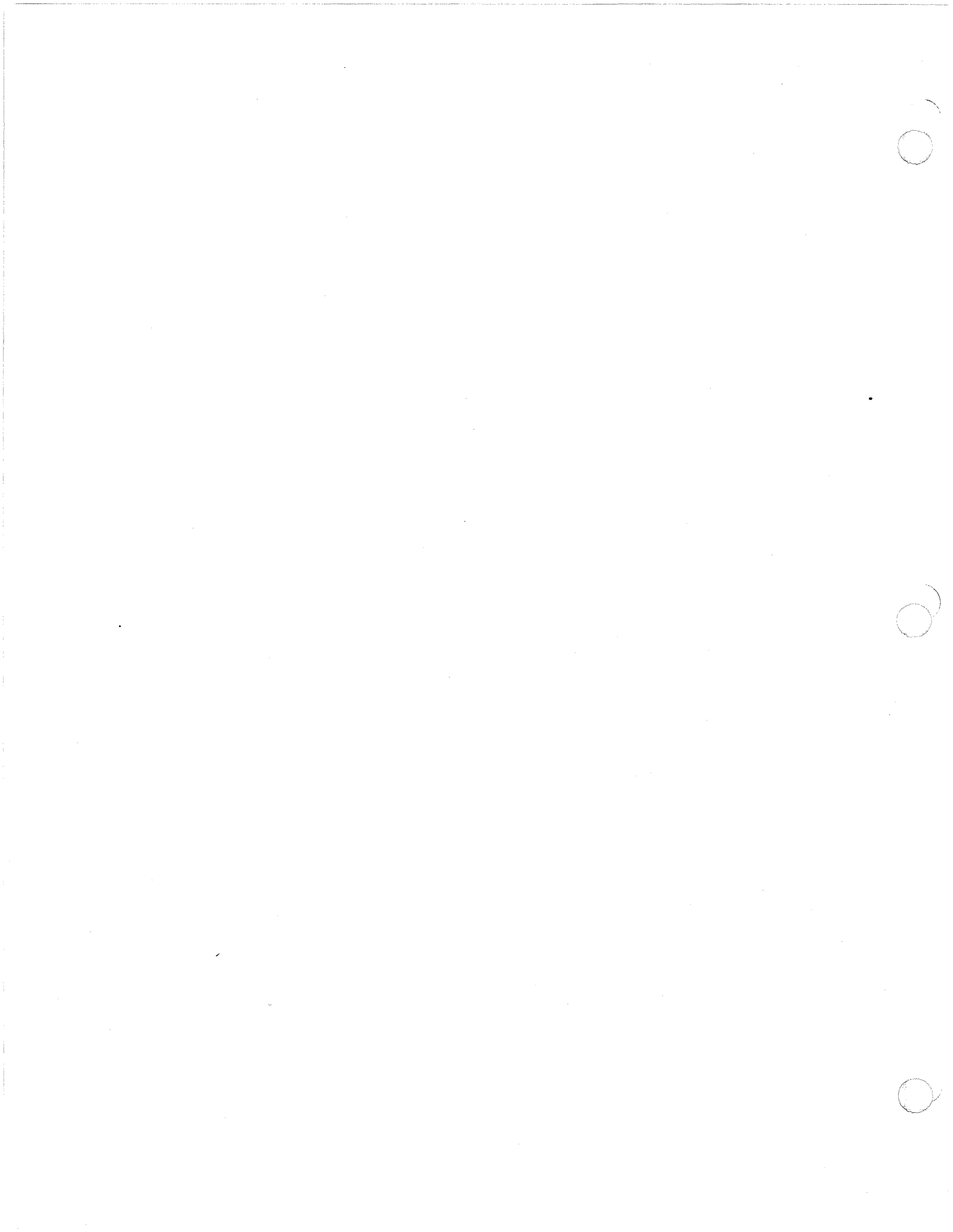
Figure 1-1. Application Development Components



REMOVE THIS PAGE AND PLACE TAB FOR :

TAB 2

SYSTEM ACCESS



## *Section 2*

# **SYSTEM ACCESS**

This section describes user access procedures and the procedures and conventions used to control the processing environment once you have accessed the system.

### USER ACCESS PROCEDURES

When you are at a user terminal, access to the system depends on the way your terminal is described to and recognized by the system. Access to the system requires:

1. Physical connection between your terminal and the central processor
2. Logical connection between you (the user) and the operating system.

In some cases, the Executive performs the second step for you automatically after you have made the physical connection.

### CONNECTING THE TERMINAL TO THE CENTRAL PROCESSOR

You can connect your terminal to the central processor by two methods, depending on the type of terminal you have: a direct-connect terminal or a dialup terminal.

## Direct-Connect Terminal

For a direct-connect terminal, place the POWER ON/OFF switch in the ON position. If the terminal has a LOCAL/ON LINE switch, place it in the ON LINE position. This is sufficient to connect the terminal to the central processor.

## Dialup Terminal

A telephone line connects a dialup terminal to the central processor. Take the following steps to make the connection:

1. Turn both the terminal POWER ON/OFF switch to the ON position and the LOCAL/ON LINE switch to the ON LINE position.
2. Lift the receiver, press the button marked TALK/CLEAR, and listen for a dial tone.
3. Use the telephone number provided by the system operator for the dialup line to call into the system.
4. Press the button marked DATA when you hear a high-pitched tone (this tone lets you know that the connection has been made). Hang up the receiver.

If you are unable to make a connection, hang up the receiver and begin again at Step 2.

## CONNECTING A USER TO THE EXECUTIVE

After you have made the physical connection between your terminal and the central processor, you can make the logical connection that identifies and establishes you as a user known to the Executive. The procedure you use depends on whether your terminal is defined as a login terminal or a non-login terminal. Login terminals are reserved for initial user access to the system through a system component called the Listener. Listener monitors all terminals that are listed in its Terminals file. Such terminals cannot be directly reserved by applications. On the other hand, non-login terminals (i.e., those not listed in the Terminals file) can be directly reserved by applications. Your system administrator can tell you what type of terminal you have.

## Login Terminal

There are two types of login terminals: manual login and direct login. Manual login terminals require you to enter a Login command line or fill out a login form in order to access the system. Direct login terminals take their Login commands from the Terminals file as soon as they are physically connected to the system; this process is invisible to the user.



## MANUAL LOGIN TERMINAL

Your terminal may or may not be configured for manual login. If it is, a login banner or form will be displayed. A manual login terminal is one that requires you to use the Login command to connect to the Executive. In order to log in to the system, you must be registered as a user. The system administrator can register you, or may allow an unregistered user to log in under the user identification USER.

Manual login terminals can be of two types: Banner Login or Forms Login. A Banner Login terminal displays a banner message and waits for you to enter a login command line. A Forms Login terminal displays a form that guides you through the login process. The following paragraphs describe the login procedure for both types of terminals.

### Banner Login

When a banner login terminal is physically connected to the system, it displays a login banner consisting of the message of the day and the login prompt:

LOGIN

followed by the terminal identification and the current date and time. You can access the system by logging in with either a full or abbreviated login line; your system administrator can tell you which type of login works on your terminal. With a full login you can specify your group id, pool id, lead task, home directory, and other login characteristics, assuming that you are not restricted from these options. (Refer to the Commands manual for a full description of the Login command, or type LOGIN ? in response to the LOGIN prompt to see a summary of LOGIN options.) If the terminal (and your user profile) allow it, you may gain access to the system by entering a full login line, such as:

```
L Jones
```

or

```
L Jones -GP JJ -HD >UDD>APPDEV>JONES>PROGS
```

In these examples, Jones is a login\_id. You may instead use your two- or three-part user\_id, such as JONES.APPDEV or JONES.APPDEV.R27.

An abbreviated login logs you in with a pre-defined or "canned" login line and allows no options. An abbreviated login line consists of a single character that has been previously established as an abbreviation for a full login line. Abbreviations can be valid at all or any banner login terminals in a system.

After typing in your login line, you may be required to enter your password in response to the prompter message:

Please enter your password.

You must then correctly enter your password. This password gives you access to the system. For security purposes, your password is not displayed as you type it in.

If this is the first time that you have logged in to the system, you may be asked (via a terminal display) to choose a password. In this case, choose a password six, seven, or eight characters long.

#### NOTE

In the event that you may want to convert to Forms Login, you should not include control characters (e.g., TAB), commercial at (@), or blanks in your password. These characters are not acceptable in Forms Login passwords.

Once you have successfully logged in to the system, what is displayed on your terminal depends on your user profile and the specifications in your login line.

If you need and are allowed to change your password, enter your login line with the -CPW argument. After you have logged in with your old password, the system displays the following message:

Choose a password six to eight characters long

After entering your new password, you must confirm it by reentering it when the following message is displayed:

Reenter the password

After you confirm your new password, the following message is displayed:

Your password has been changed.

#### Forms Login

When a forms login terminal is physically connected to the system, it displays the form shown in Figure 2-1. For a normal login using Forms Login, you must enter your login id, press the RETURN key, enter your password (if required by the System Administrator), and press the XMIT key.

```

Welcome to your Honeywell System.

ID: _____

PASSWORD:

OPTIONS: To change your password, type C.
         To override your login defaults, type O. _
```

Figure 2-1. Login Form\

If this is the first time that you have logged in to the system, you may be asked (via a terminal display) to choose a password. In this case, choose a password six, seven, or eight characters long, which doesn't include any of the following:

- Control characters (e.g., TAB character)
- Commercial at (@)
- Leading or embedded blanks (i.e., a blank with another character following it)

Once you have successfully logged in to the system, what is displayed on your terminal depends on your user profile and the specifications in your login line.

If you are allowed to change your password, you may type C in the last field of the form shown in Figure 2-1 before pressing XMIT. The following message will appear on the blanked-out screen:

```
Choose a password six to eight characters long
```

After entering your new password, you must confirm it by reentering it when the following message is displayed:

```
Reenter the password
```

After confirming your new password, the following message is displayed:

```
Your password has been changed.
```

To specify any of the optional login arguments that may follow your id on the login line, type O in the last field of the form shown in Figure 2-1 and press the XMIT key. The form shown in Figure 2-2 will be displayed. Make the appropriate entries in the fields for any arguments you want to specify and press the XMIT key.

TEMPORARY OVERRIDES OF YOUR LOGIN DEFAULTS:  
 Any entries you make will change your defaults for this login only.  
 If you make no entries, you will be logged in with your defaults.

```

  / Distination group for secondary login:  ___ \ You may skip these fields
  \ Group-id for primary login:           ___ / or enter either, but not both.
Home directory: _____
Lead task pathname: _____
Hold phone line on logout? (Y/N)  N
Memory pool: _____
Number of LRNs:  ___           Number of LFNs:  ___
Number of IRBs:  ___           Number of TSAs:  ___
Relative priority level: _____ Language key:  ___
Arguments to lead task (start on first line):
_____
_____
_____
  
```

Figure 2-2. Login Arguments Form

### DIRECT LOGIN TERMINAL

Your terminal may or may not be configured for direct login. If it is, there is no login prompt. The login process occurs automatically after connecting to the central processor. After the message of the day (if there is one), the system will respond with either a ready message or a menu.

### Non-Login Terminal

A non-login terminal does not require a user Login command, either manually (from the terminal) or directly (from the Terminals file). When this terminal is physically connected to the system, it does not display a login banner or form. At the administrator's option, it may display the message of the day. Usually you can start entering data immediately after making a logical connection with the system. What is displayed on the terminal depends on the characteristics of the lead task of the task group associated with the terminal.

If your lead task is the command processor, it is recommended that the first command you enter be the Ready On (RDN) command, if it is not already included in your START UP.EC file. If the system responds with the Ready prompt (RDY:), you can continue with the session. If the system does not respond either to this command or any other command, you can request the operator to activate a task group for that terminal. After the task group is activated, your access to system facilities is governed by the control arguments specified in the task group activation command.

Example:

You have made a connection with the system. Your task group has the command processor as its lead task. The first user command you enter is:

**RDN**

to activate the ready prompt message. The system responds with:

RDY:

Enter the command to list your working directory:

**LWD**

The system responds (in this case) with:

^ZSYS51>SOURCE>CODE  
RDY:

To find out what files are in your working directory, enter:

**LS**

and the system responds with a listing of the files. Figure 2-3 shows a sample directory listing.

```
DIRECTORY: ^ZSYS51>SOURCE>CODE  
  
TIME: 1986/05/04 0724:27  
  
ORDIN.C          SEQ      8  
ORDOUT.C         SEQ      8  
PAY.C            SEQ      8  
INVNTY.C         SEQ      8  
  
Total Sectors:  32
```

Figure 2-3. Sample Directory Listing

## PROCEDURES AND CONVENTIONS AFTER ACCESS

Once you have successfully accessed the system, what is displayed on your terminal depends on the task group associated with your terminal. When the terminal is a login terminal, Listener connects your terminal to either a task group that is spawned according to your user registration information or an existing task group as a secondary terminal. If you are a command-mode user and the Ready On command is specified in your START\_UP.EC, the system ready prompt is displayed. If you are a

\* Menu Subsystem user, a menu is displayed.

### \* Sending Messages to the Operator

It may be necessary at times to request operator intervention while at your terminal. To send a message from your terminal to the system operator (e.g., when the terminal is remote from the operator terminal), enter the Message (MSG) command. Messages sent to the operator are displayed immediately. For example, to send a message to abort the current batch request, you might enter:

```
MSG "PLEASE ABORT BATCH REQUEST"
```

If your message contains embedded blanks, you must enclose it in quotation marks or apostrophes.

### Interrupting a Task

It may be necessary at times to interrupt a running task while at your terminal. You can interrupt or break a running task in order to reenter commands, temporarily halt, or terminate the task. The break function immediately interrupts the command currently being processed and returns control to command level for further action.

To effect a break from your terminal, press the BREAK (BRK) key (or the key corresponding to the BREAK key on your terminal). The system then issues the break message:

```
**BREAK**
```

If your terminal was processing a form or menu, acknowledge the break message that is displayed on the supervisory line.

Your response to the break can be any one of the following:

1. Enter any user command (see the Commands manual). If the entered command is not Start (SR), Logoff (BYE), New Process (NEW\_PROC), Unwind (UW), or Program Interrupt (PI), the lead task again enters the break mode and issues another **\*\*BREAK\*\*** message, requesting another response. This can be followed by another user command or by one of the responses described in step 2. If your terminal was processing a form or menu, acknowledge each message that is displayed on the supervisory line.
2. Enter one of the following break mode responses to the **\*\*BREAK\*\*** message:
  - a. Start (SR): This resumes execution of the suspended task as though the break had not been made. It also terminates the current break; i.e., there is no other **\*\*BREAK\*\*** message after it executes.
  - b. Unwind (UW): This terminates the current task, and you return to command level. If the terminated task was invoked following a break, the lead task reenters the break mode, issues another **\*\*BREAK\*\*** message, and awaits a response.
  - c. Logoff (BYE): This aborts and deletes the current task group request.
  - d. New Process (NEW\_PROC): This aborts all task requests in the task group except for the lead task, then restarts the task group, using the same arguments as specified in the initial task group request. It also terminates the current break; i.e., there is no other **\*\*BREAK\*\*** message after it executes.
  - e. Program Interrupt (PI). The interrupted task is currently suspended. The PI command is meaningful only to the Linker and Editor running in a task group whose lead task is the Command Processor. For Linker and Editor, the command suppresses output and returns to directive input level. The PI command suppresses output resulting only from the Linker MAP directive.

If your terminal was processing a form or menu, acknowledge each message that is displayed on the supervisory line.

**Example:**

You issue a List Names (LS) command and the output begins to appear on the screen at your terminal. You wanted this output to be printed on the line printer. You should immediately press the BREAK key and take one of the following steps:

1. Enter:

```
FO !LPT00
```

to change the output destination to the line printer; then enter the SR command to resume execution of the LS command. The output that had already appeared at the terminal does not appear on the hard-copy printout.

2. Enter the UW command to terminate the current LS task, and enter:

```
FO !LPT00
```

to change the output destination to the line printer. Then enter the LS command to restart the LS program from the beginning.



REMOVE THIS PAGE AND PLACE TAB FOR

TAB 3

FILE CONVENTIONS



## *Section 3*

# **FILE CONVENTIONS**

This section presents MOD 400 file conventions as well as a procedural scenario titled "Working With Files." This scenario provides a detailed explanation of frequently used file system commands and procedures.

### OVERVIEW

A file is a logical unit of data composed of a collection of records. The principal external devices available for storing files are:

- Disk devices (diskettes, cartridge disks, cartridge module disks, and mass storage units)
- Magnetic tape units.

These external devices are referred to as volumes (e.g., diskette volume, tape volume).

Various conventions to identify and locate files stored on disk and magnetic tape have been established for their effective control. The conventions facilitate the orderly and efficient use of the stored data.

Unit record devices (such as card readers and printers) also use the file concept. However, since unit record devices cannot be used to store files, there is less need to establish conventions for identification and location. A unit record file is simply the data that is read or written at any one time.

## DISK FILE CONVENTIONS

Users must be able to specify an access path to any given file on a disk volume that contains multiple files. Files must, therefore, be organized on the volume in some predictable fashion. MOD 400 provides a set of volume organization conventions by which the system can locate any element that resides on the volume.

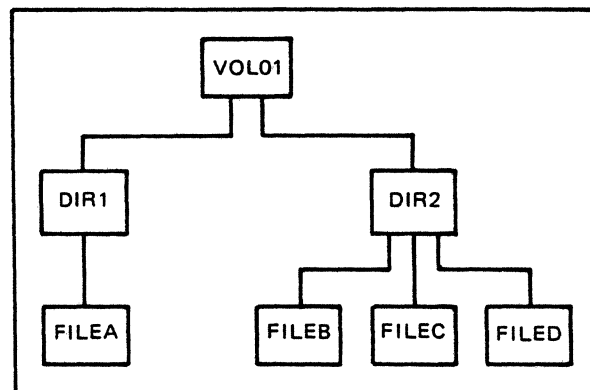
The principal elements of this organization, aside from the files themselves, are directories. The access path to any given element on a volume is known as a pathname.

### Directories

Files on disk devices reside within a tree-structured hierarchy. The basic elements of this hierarchy are files known as directories. The directories are used to point to the location of data files, which are the endpoints of the tree structure.

A directory on a disk volume functions like a catalog. It contains the names and starting locations (sectors on the volume) of files or other directories (or both). The elements whose names are in the directory are said to be contained in or subordinate to the directory; therefore, the organization of a disk volume is a multilevel structure. The complexity of the access path to any given element in the structure depends on the number of directories between the root and the desired element.

A directory structure is illustrated in Figure 3-1. The base directory on a volume is termed a root directory. In Figure 3-1 the root directory is VOL01. The root directory VOL01 points to two subordinate directories DIR1 and DIR2. The directories DIR1 and DIR2, in turn, point to the data files (FILEA, FILEB, FILEC, and FILED).



84-817

Figure 3-1. Example of Disk File Directory Structure

The following paragraphs describe the root directory and other special types of directories.

#### ROOT DIRECTORY

There is a tree structure for each disk mounted at any given time. At the base of each tree structure is a directory known as the root directory. This is the directory that ultimately contains every element that resides on the volume either immediately or indirectly subordinate to it.

The root directory name is the same as the volume identifier of the volume on which it resides. The directory VOL01 in Figure 3-1 is a root directory.

#### SYSTEM ROOT DIRECTORY

One or more disk root directories can be known to the system at any time during its operation. One of these, the system root directory, is required at all times. The volume used by the operator to initialize the system establishes the system root directory. This volume also normally contains system programs, commands, and other routinely used elements. It must contain a number of directories and files that the system needs to perform its functions. These are described in the System Building and Administration manual.

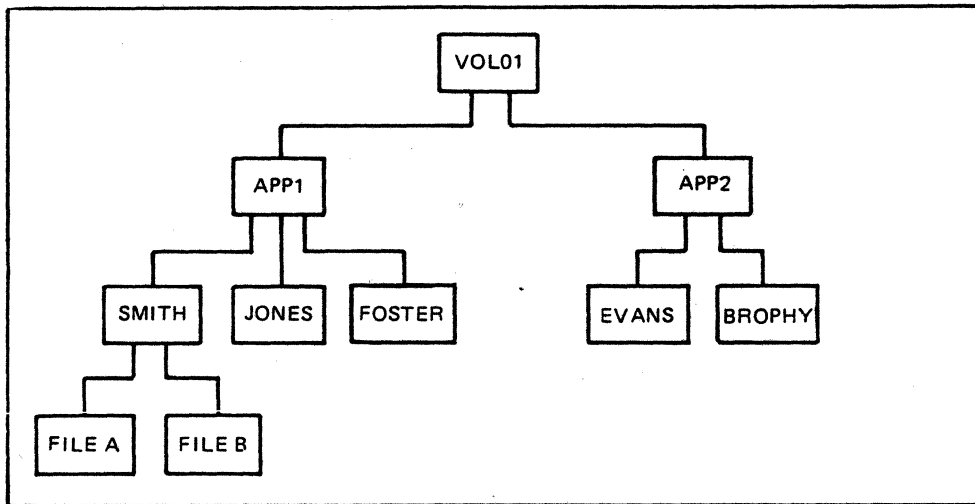
#### USER ROOT DIRECTORIES

The File System can recognize one or more user root directories. These are root directories of volumes created and used for the installation's own particular needs. They can contain user application programs and their associated data files, application program source and object unit files, listing files, or anything else to be stored, either temporarily or permanently.

#### INTERMEDIATE DIRECTORIES

When a volume is first created, it contains only a root directory. Within this directory, you can create any additional directories required to satisfy the needs of the installation. Consider, for example, a volume that is to contain data used by two application projects, each of which has several people associated with it. Each of these people has one or more files of interest to him. The volume has been initialized and contains a root directory name. Two directories can be created subordinate to the root directory, each identified by the project name. Then, subordinate to these directories, a directory can be created for each person associated with each project.

The data files are all contained within the personal directories. This sample intermediate directory structure is illustrated in Figure 3-2.



84-818

Figure 3-2. Sample Directory Structure

When the need for a user-created directory no longer exists, the directory can be deleted from the File System, making the space it occupied, as well as the space occupied by its attributes in the immediately superior directory, available for reuse. A directory must be empty before it can be deleted; all directories and files subordinate to the one to be deleted must have been previously deleted by explicit commands.

#### WORKING DIRECTORY

The File System always starts at a root directory when it performs an operation on a disk file or a directory. At times the search for an element residing on a disk volume can traverse a number of intermediate directory levels before locating the desired element; the File System must be supplied with the names of all the branch points it must pass on the way. The files of interest to a user doing work on the system are frequently all contained in a single directory specific to the task being performed; this directory can be three, four, or more levels deep into the structure. It would be convenient to be able to refer to files in relation to a directory at some arbitrary level in the hierarchy rather than in relation to the root directory. The File System allows this to be done by recognizing a special kind of directory known as a working directory.

A working directory establishes a reference point that enables you to specify the name of a file or another directory in terms of its position relative to that directory. If the access path of the working directory is made known to the File System, and if the desired element is contained in that directory, the element can be specified by just its name. The File System concatenates this name with the names of the elements of the working directory's access path to form the complete access path to the element.

## LOCATIONS OF DISK DIRECTORIES AND FILES

The File System has total control over the physical location of space allocated to directories and files; you need never be concerned about where on a volume a directory or file resides. When a volume is first initialized, space is allocated to elements in essentially the order in which they are created. But after the volume has been in use for some time, elements may have been deleted and the space they occupied made reusable. Hence, when a new element is created, it is allocated the first available space even though that space may eventually be too small to contain the file. If more space is needed for even a single extent of a file, it will be obtained from another free area. Thus, there is not necessarily any relationship between a file's extents and contiguous free disk sectors.

### Naming Conventions

Each disk file and directory name in the File System can consist of the following ASCII characters: uppercase alphabetic (A through Z), lowercase alphabetic (a through z), digits (0 through 9), underscore (\_), apostrophe ('), hyphen (-), period (.), and dollar sign (\$). On systems that use an 8-bit ASCII Extended Character Set (80 through FF hexadecimal), the disk file and directory names can also include the international graphic characters in the C0 through FF hexadecimal range. File names are stored on the disk media and will be displayed as they were created. In any reference to a file name, lowercase alphabetic are treated the same as uppercase alphabetic and lowercase extended letters (E0 through FF hexadecimal) are treated the same as uppercase extended characters (C0 through DF hexadecimal). For example, "FILE", "File", or "file" are equivalent and could be supplied in pathname requests to reference the same file.

The first character of any name must not be hexadecimal FF (lowercase y with diaeresis) or hexadecimal 2E (period). The underscore can be used to join two or more words that are to be interpreted as a single name (e.g., DATE\_TIME). A period followed by one or more alphabetic or numeric characters after a file name is normally interpreted as a suffix to a file name. This convention is followed, for example, by a compiler when it generates a file that is to be listed; the compiler identifies this file by creating a name of the form FILE.L.

The name of a root directory or a volume identifier can consist of from one to six characters. The names of other directories and files can comprise from 1 to 12 characters. The length of a file name must be such that any system-supplied suffix does not result in a name of more than 12 characters.

#### UNIQUENESS OF NAMES

Within the system at any given time, the access path to every element must be unique. This leads to the following rules:

- Only one volume with a given volume\_id can be mounted at any given time. (The system informs you of an attempt to mount a volume having the same name as one already mounted.)
- Within a given directory, every immediately subordinate directory name must be unique. (The Create Directory command informs you of an attempt to add a duplicate directory name.)
- Within a given directory, every file name must be unique. (The Create File command informs you of an attempt to add a duplicate file name.)

#### PATHNAME

The access path to any File System entity (directory or file) begins with a root directory name and proceeds through zero or more subdirectory levels to the desired entity. The series of directory names (and a file name if a file is the target entity) is known as the entity's pathname. The total length of any pathname, including all hierarchical symbols, cannot exceed 57 characters, except that a working directory pathname cannot exceed 51 characters.

#### Symbols Used in Pathnames

The following symbols are used to construct pathnames.

- Circumflex (^)--Used exclusively to identify the name of a disk volume root directory. The circumflex is used in two forms. In one form it directly precedes the root directory name (e.g., ^VOL011). In the other it directly precedes a greater-than symbol (>) to refer to the root directory of the current working directory (e.g., ^>DIR1>FILEA).



- Greater than (>)--Indicates movement in the hierarchy away from the root directory. The symbol is used to connect two directory names or a directory name and a file name. It can also be the first character of a pathname, in which case the element whose name follows it is immediately subordinate to the root directory of the system volume. Each occurrence of the greater-than (>) symbol denotes a change of one hierarchical level; the name to the right of the symbol is immediately subordinate to the name on the left. Reading a pathname from left to right thus indicates movement through the tree structure in a direction away from the root directory. If the root directory ^VOL011 contains a directory name DIR1, the pathname of DIR1 is:

```
^VOL011>DIR1
```

If the directory named DIR1 in turn contains a file named FILEA, then the pathname of FILEA is:

```
^VOL011>DIR1>FILEA
```

The greater-than symbol (>) is never followed by a space. Pathnames that begin with two greater than symbols (>>) assume that the entities specified are subordinate to the system boot directory.

- Less than (<)--Used at the beginning of a pathname to indicate movement from the working directory in a direction toward the root directory. Consecutive symbols can be used to indicate changes of more than one level; each occurrence represents a one level change. When followed by elements of a relative pathname, those elements represent changes of direction away from the root directory. One or more of these symbols may precede only a relative pathname.
- ASCII space character--Used to indicate the end of a pathname. When represented in memory, a pathname must end with a space character.

The last (or only) element in a pathname is the name of the entity upon which action is to be taken. This element can be a device name, directory name, or file name, depending on the function to be performed. In the Create Directory command, for example, a pathname specifies the name of a directory to be created. The last element of this pathname is interpreted by the command as a directory name; any names preceding the final name are names of superior directories leading to it. An analogous situation occurs with the Create File command, except that in this case the final pathname element is the name of a file to be created.

## Absolute and Relative Pathnames

A full pathname contains all necessary elements to describe a unique access path to a File System entity, regardless of the type and location of the device on which it resides. The File System uses this form in referring to a directory or file. However, it is frequently unnecessary to specify all of these elements; the File System can supply some of them when the missing elements are known to it and the abbreviated pathnames are used in the appropriate context. An understanding of these conditions and contexts requires an understanding of absolute and relative pathnames.

### Absolute Pathname

An absolute pathname is one that begins with a circumflex (^) or a greater-than (>) symbol. A pathname that begins with a circumflex is a full pathname. A pathname that begins with either a single greater-than symbol or two greater-than symbols identifies the root directory of the system root volume or the system boot volume. Thus, if the system root volume name is SYS01 and the pathname given is >DIR1>FILEA, the full pathname becomes ^SYS01>DIR1>FILEA. Using the greater-than symbol, you don't have to know the names of the system volumes. In addition, the name of the root directory of your current working directory can be abbreviated to ^>. That is, ^>DIR2>FILEB references another file on the same volume as the working directory.

Another volume, USER1, can also contain a >DIR1>FILEA access path and can be known to the File System; the two access paths are made unique by requiring that the root directory be specified when referring to the second volume. The full pathname of this file on the second volume is thus ^USER1>DIR1>FILEA.

### Relative Pathname

A relative pathname is a shortened version of the absolute pathname and assumes the working directory (or a higher element in the directory structure) without explicitly referring to it in the pathname. A relative pathname can begin either with a file or directory name or with one or more less-than symbols (<). If the pathname begins with a name (e.g., DIR1>FILEA or FILEA), the elements so identified are immediately subordinate to the working directory. If a relative pathname begins with a less-than symbol (e.g., <FOSTER), the name following the less-than symbol identifies an element that is immediately subordinate, not to the working directory but to the element to which the working directory is itself immediately subordinate. Thus, the less-than symbol is a way of working back up the directory structure.

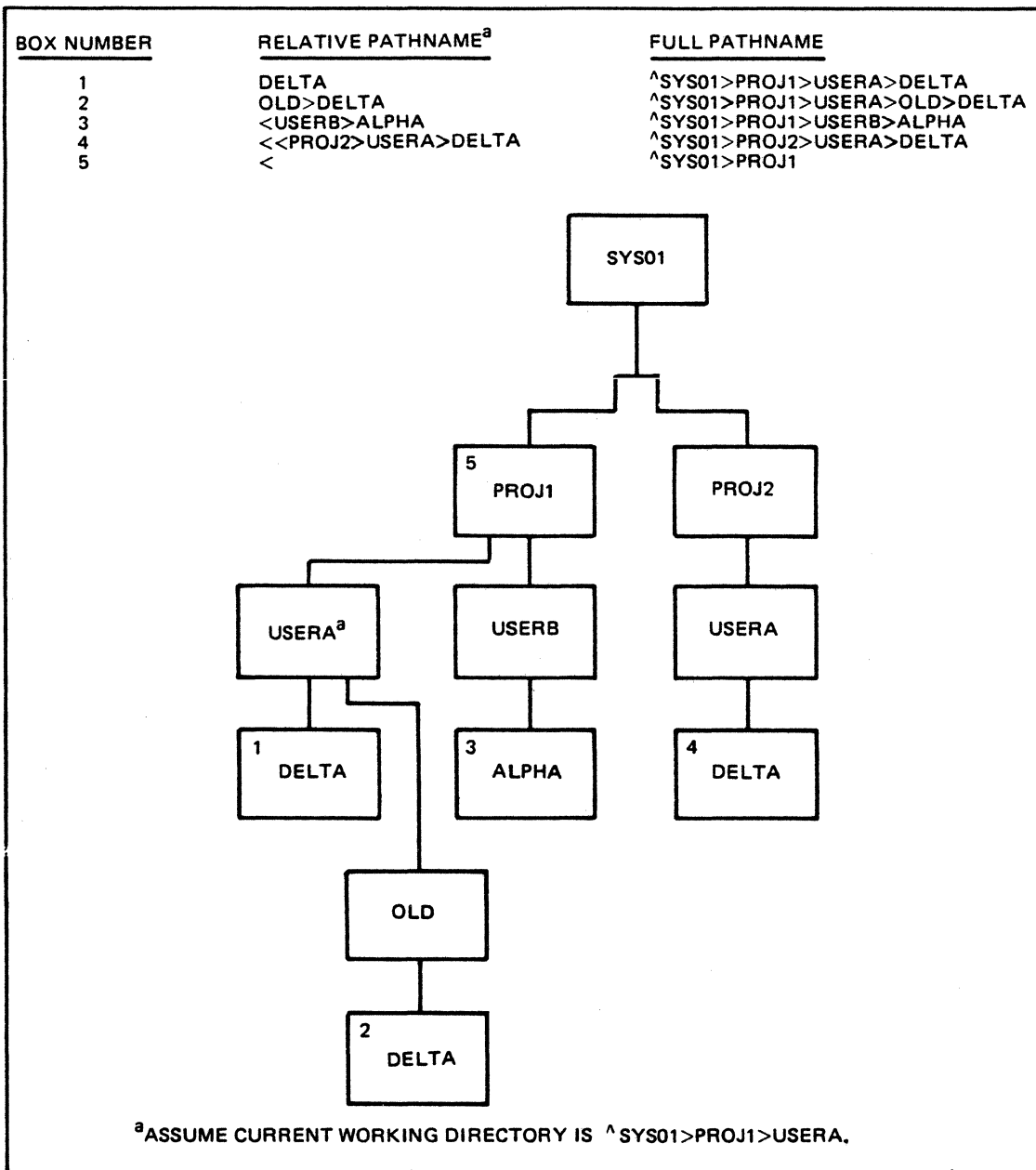
A relative pathname can consist of one or more elements. If a relative pathname contains more than one element, each element except the last must be a directory name, the first immediately subordinate to the current working directory level, the second immediately subordinate to the first, and so on. The last or only element can be either a directory name or a file name, depending on the function being performed, as described previously.

A simple name is a special case of the relative pathname. It consists of only one element: the name of the desired entry in the working directory.

If a reference is to be made to a file or directory that is on the same volume but not subordinate to the working directory, there are two alternative ways of making this reference: by using an absolute pathname, or by using any of the forms of relative pathname described previously.

Figure 3-3 shows some relative pathnames and the full pathnames they represent when the working directory pathname is:

```
>PROJ1>USERA
```



84-819

Figure 3-3. Sample Pathnames

## MAGNETIC TAPE FILE CONVENTIONS

The magnetic tape file conventions include tape file organization, tape file naming conventions, and tape file pathnames.

### 9-Track Magnetic Tape File Organization

9-track magnetic tape supports only the sequential file organization. Fixed- or variable-length records can be used. Records cannot be inserted, deleted, or modified, but they can be appended to the file. The tape can be positioned forward or backward any number of records.

The unit of transfer between memory and a tape file is a block. Block size varies depending on the number of records and whether the records are fixed or variable in length.

A block can be treated as one logical record called an "undefined" record. An undefined record is read or written without being blocked, unblocked, or otherwise altered by data management. Spanned records (i.e., those that span two or more blocks) are supported. (No record positioning is allowed with spanned records.)

A labeled tape is one that conforms to the current tape standard for volume and file labels issued by the American National Standard Institute. The following types of labeled tapes are supported:

- Single-volume, single-file
- Multivolume, single-file
- Single-volume, multifile
- Multivolume, multifile.

The following types of unlabeled tapes are supported:

- Single-volume, single-file
- Single-volume, multifile.

### Magnetic Tape File and Volume Names

Each tape file and volume name in the File System can consist of the following ASCII characters:

- Uppercase alphabetic (A through Z)
- Exclamation mark (!)
- Double quotation marks (")
- Percent sign (%)
- Ampersand (&)
- Single quotation mark (')
- Left parenthesis ((
- Right parenthesis ())
- Asterisk (\*)
- Plus sign (+)

- Comma (,)
- Hyphen (-)
- Period (.)
- Slash (/)
- Colon (:)
- Semicolon (;)
- Less-than sign (<)
- Dollar sign (\$)
- Equal sign (=)
- Question mark (?)
- Underscore (\_).

Any of these characters can be used as the first character of a file or volume name. The underscore ( \_ ) can be used as a substitute for a space. If a lowercase alphabetic character is used, it is converted to its uppercase counterpart.

The name of a tape volume can be from one through six characters; tape file names can be from 1 through 17 characters.

### Magnetic Tape Device Pathname Construction

As previously mentioned, magnetic tape volumes can be labeled or unlabeled.

#### UNLABELED TAPE PATHNAMES

A tape device pathname must always be used when referring to an unlabeled tape. The general form of a tape device file pathname is:

```
!dev_name
```

where dev\_name is the symbolic name defined for the tape device at system building time.

#### LABELED TAPE PATHNAMES

Labeled tapes can be referenced either by the tape device pathname convention or by the tape volume id convention. The tape device file pathname convention is:

```
!dev_name >vol_id[>filename]
```

where dev name is the name of the tape device as specified at system building time, vol\_id is the name of the tape volume, and filename is the name of the file on the volume. This convention requires that the volume be mounted on the specified device.

The tape volume id convention is:

`^vol_id[>filename]`

where vol id is the name of the tape volume and filename is the name of the file on the volume. This convention allows the volume to be mounted on any available tape device.

#### Automatic Tape Volume Recognition

Automatic volume recognition dynamically notes the mounting of a tape volume. This feature allows the File System to record the volume identification in a device table, thus making every tape volume accessible to the File System software.

#### UNIT-RECORD DEVICE FILE CONVENTIONS

Unit-record devices (e.g., card readers, card punches, printers) are used only for reading/writing data; they are not used for data storage and thus do not require conventions for file identification and location.





Refer to a unit-record device by entering a pathname consisting of an exclamation mark (!) followed by the symbolic device name defined during system building. The format is:

!dev\_name

where dev\_name is the symbolic device name of the unit record device.

### WORKING WITH FILES

The following information addresses selected commands and procedures that you can use frequently, including:

- Using file pathname conventions
- Controlling your files and directories
- Interrupting execution
- Controlling your output
- Controlling printing
- Program execution
- Communicating with other users
- Performing batch processing.

The examples that follow provide full details on performing these functions. Note that some examples do not list all optional control arguments for the commands described. See the Commands manual for a complete description of all commands and their arguments.

### COMMAND PROCESSOR

You communicate with the system through command lines entered at a terminal or read from a command file. Command lines are read and interpreted by a system software component called the Command Processor.

#### Standard I/O Files

Four files are always associated with the Command Processor:

- Command-in
- User-in
- User-out
- Error-out.

The command-in file is the file from which the Command Processor takes its input. The command-in file is normally associated with (or assigned to) your terminal. However, it can be reassigned, temporarily, to another device or file, and subsequently reassigned to the terminal.

A command function reads its own input during execution from the user-in file (normally assigned to your terminal). The directives submitted to the Editor following entry of the Editor command, for example, are submitted through user-in. A task group normally writes its output to the user-out file (normally assigned to your terminal). The user-out file can be reassigned to another device or file (see "Controlling Output"). This reassignment remains in effect until another reassignment occurs.

The Command Processor, and any commands it invokes, writes any errors detected to the error-out file. The initial error-out file is the same as the initial user-out file; it can be reassigned by the File Out (FO) command.

Full pathnames associated with each of these files can be determined by issuing a Status Group (STG) command at the terminal.

### Command Level

The system indicates that it is at command level by issuing a ready (RDY) message at your terminal. This assumes that you have not disabled the ready message by a previously issued Ready Off (RDF) command; if you have, the system still comes to command level, but you are not informed. You can activate the ready prompt at any time by issuing a Ready On (RDN) command.

When executing a command function, you can return to command level in one of two ways:

- After a command function terminates, the system returns to command level and awaits the entry of another command. This command can be any function you wish to execute or it can be a BYE command, indicating that you have no further work to do and you want to terminate the current session.
- You can interrupt execution of an invoked command by pressing the Break or Interrupt key at your terminal. See "Interrupting Execution" below.

### CONTROLLING YOUR OPERATING ENVIRONMENT

The following paragraphs describe the commands and procedures that may be most useful to an interactive system user. Once at command level, a wide variety of system operations can be performed using these commands and special system procedures. Selected examples are designed to help you become familiar with using the system for applications programming. For full descriptions of all commands and their arguments, refer to the Commands manual.

## Volume Control

The following commands illustrate how to create or rename a disk volume. \*

### CREATING VOLUMES

Before beginning useful work on a previously unused tape or disk volume, assign it a unique name (volume identifier or vol\_id) that can be recognized by the system. The vol\_id designates the volume (or root) directory name of the tape or disk volume.

Ask the system operator to mount your diskette or tape on an available drive and notify you of the drive's symbolic peripheral device name. (The symbolic peripheral device name is the name the system uses to recognize the device.)

For example, suppose you want to create a diskette volume and name it WORK. Send the following message to the system operator (see "Communicating With Other Users" later in this section):

```
MSG "MOUNT DISKETTE AND NOTIFY ME OF DEVICE NAME"
```

The operator issues the Status System (STS) operator command to determine which devices are available:

```
STS
```

and the system responds:

|   | SYMPD<br>NAME | CHANNEL | DEVICE<br>TYPE | VOLUME<br>ID | USAGE | AVAILABLE<br>PHYSICAL | SECTORS<br>LOGICAL | VOL/FILE<br>SET NAME | MEMBER<br>NUMBER |
|---|---------------|---------|----------------|--------------|-------|-----------------------|--------------------|----------------------|------------------|
| B | RCM00         | 2800    | 2380           | DMPVL        | 0     | 46504                 | 5813               |                      |                  |
| B | RCM01         | 2880    | 2380           | OPEN         | 0     | 51704                 | 6463               |                      |                  |
| B | FCM01         | 2880    | 2385           | SYSTST       | 0     | 172792                | 21599              |                      |                  |
| B | MSM01         | 1880    | 2361           |              | D     |                       |                    |                      |                  |
| B | RCD00         | 1400    | 2332           | MINE         | 0     | 9465                  | 1182               |                      |                  |
| B | FCD00         | 1400    | 2333           |              | 0     | 0                     | 0                  |                      |                  |
| B | RCD01         | 1480    | 2332           | RJE          | 0     | 1216                  | 152                |                      |                  |
| B | FCD01         | 1480    | 2333           | FCD01        | 0     | 19560                 | 2445               |                      |                  |
| B | FCD02         | 1500    | 2333           | FCD02        | 0     | 11960                 | 1495               |                      |                  |
| B | RCD03         | 1580    | 2332           |              | D     |                       |                    |                      |                  |
| B | FCD03         | 1580    | 2333           |              | D     |                       |                    |                      |                  |
| B | DSK00         | 0400    | 2010           |              | D     |                       |                    |                      |                  |

The operator then mounts a diskette on the available drive, DSK00, and sends you the following message:

```
VOLUME MOUNTED ON DSK00
```

You can now use the Create Volume (CV) command to assign a unique vol\_id to your new disk volume, using the following form of the command:

```
CV !DSK00 -FT WORK
```

where WORK is the vol\_id you want to assign.

Using the -FT argument initializes all data structures on the volume and establishes WORK as the root directory name; the root directory pathname for this volume is ^WORK. Since this command will erase any data on the disk, make certain you use the correct device name.

#### RENAMING DISK VOLUMES

If disk volumes having the same vol\_id are used, one of the volumes must be renamed before the system will accept it. (A tape volume cannot be renamed.) The command:

```
CV !DSK00>OLD -RN NEW
```

renames the volume OLD using the -RN control argument; the new volume name is NEW.

#### Directory Control

You can create an unlimited number of directories to organize your files. The following commands illustrate how to change your working directory, and create, rename, or delete directories.

#### CHANGING YOUR WORKING DIRECTORY

The system provides you with tools to keep you aware of your location within the directory and file structure at any moment. You can also request a list of the files and directories under any directory to which you have list access.

To list your working directory, use the List Working Directory (LWD) command:

```
LWD  
^SYSVLA>UDD>PROGS>LOWELL
```

The system responds with the absolute pathname of your working directory. If you want to change to some other directory, use the CWD (Change Working Directory) command. For example:

```
CWD ^SYSVLA>UDD>PROGS>JONES  
RDY:  
LWD  
^SYSVLA>UDD>PROGS>JONES
```

The name of your new working directory is JONES. Any number of users can work in the same directory at one time, as long as each user has list access to move there.

It is usually more convenient to use the relative pathnames of directories. For example, you can change your working directory to LOWELL by typing:

```
CWD <LOWELL
```

When going from a directory to a subdirectory, the system requires you to specify the directory name (there may be more than one directory subordinate to your working directory).

However, when moving up in the file structure, there is no ambiguity. You can move up one or more directory levels by entering one or more "<" signs:

```
CWD <  
RDY:  
LWD  
^SYSVLA>UDD>PROGS
```

If CWD is entered with no arguments, the system returns you to your home directory (your initial working directory):

```
LWD  
^SYSVLA>UDD>PROGS>JONES  
RDY:  
CWD  
RDY:  
LWD  
^SYSVLA>UDD>PROGS>LOWELL
```

## CREATING DIRECTORIES

You can create a directory using the Create Directory (CD) command. For example, you may want to put a COBOL program and a BASIC program under separate subdirectories below your home directory (your initial working directory). You first create the directories:

```
CD COBOL_DIR  
RDY:  
CD BASIC_DIR
```

You can now create your programs in subdirectories subordinate to your home directory (or create them elsewhere and copy them into the directories COBOL\_DIR and BASIC\_DIR).

As another example, suppose that you have just created, formatted, and named the disk volume WORK, as described under "Creating Volumes." You would like to create two directories, named SHEPARD and COOK, immediately subordinate to the root directory ^WORK.

Before creating your two directories, you enter a CWD command to change your working directory to ^WORK:

**CWD ^WORK**

(Note that this step is optional; you need not change your working directory to the volume ^WORK to create subordinate directories or files. You can create directories or files from any location in the File System tree structure by supplying the appropriate absolute or relative pathname of the file or directory you wish to create. However, for the sake of simplicity, only simple pathnames are used here.)

To create the directory SHEPARD, enter the command:

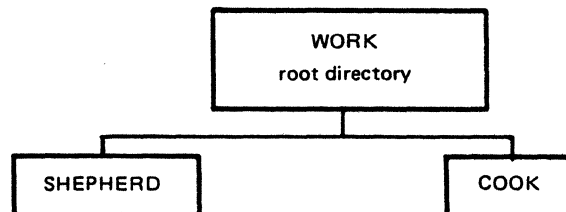
**CD SHEPARD**

This directory now resides immediately subordinate to the root directory ^WORK.

To create the directory COOK, enter the command:

**CD COOK**

This directory now resides, along with SHEPARD, immediately subordinate to the root directory ^WORK. Figure 3-4 illustrates this directory tree structure.



86-049

Figure 3-4. Location of Directories SHEPARD and COOK

## RENAMING DIRECTORIES

You can change the name of an existing directory using the Rename (RN) command. For example, assume that within your working directory >UDD>PROGS>SMITH, there is a directory TEST. The command:

**RN TEST WORK**

changes the pathname of the affected directory from:

>UDD>PROGS>SMITH>TEST to >UDD>PROGS>SMITH>WORK

## DELETING DIRECTORIES

You can delete one or more directories using the Delete Directory (DD) command. For example, you may no longer need to use a directory called EXAMPLE. The command:

**DD EXAMPLE**

deletes the directory called EXAMPLE from your working directory. Note that you could not delete EXAMPLE if it was your working directory.

As a safety measure, the File System will not allow you to delete a nonempty directory. If you wish to delete a directory, you must first delete any subdirectories or files it contains.

## File Control

The following commands show you how to create, rename, delete, copy, and locate files.

## CREATING FILES

You create a file in the file structure with the Create File (CR) command. For example:

**CR DATAFILE**

produces a sequential file called DATAFILE in your working directory, with a record size of 216 characters (the default) and a length of zero sectors. The following command:

**CR MIME.D -RSZ 80 -MSZ 800**

produces a file called MIME.D in the working directory, with a record size of 80 characters and a maximum allowable size of 800 control intervals. This file is meant to be a card file; after reading the cards (see "Copying Files" later in this section), a listing reveals the following:

**LS -LG**

DIRECTORY: ^SYSVL1>UDD>PROGS>DIRA

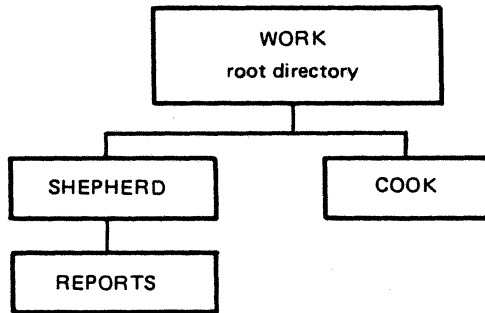
| ENTRY NAME  | TYPE | PHYSICAL SECTORS | STARTING SECTOR HEX | RECORD LENGTH |
|-------------|------|------------------|---------------------|---------------|
| START UP.EC | S    | 8                | 580                 | 256           |
| MIME.D      | S    | 40               | 800                 | 80            |

As another example, assume that you wish to create a file under each of the two directories, SHEPARD and COOK, shown in Figure 3-5. Your working directory is the root directory WORK. To create a file named REPORTS under the directory SHEPARD, enter the command:

**CR SHEPARD>REPORTS**

where SHEPARD>REPORTS is the relative pathname (relative to your working directory) of the file you wish to create.

The file REPORTS now resides immediately subordinate to the directory SHEPARD, as shown in Figure 3-5.



86-050

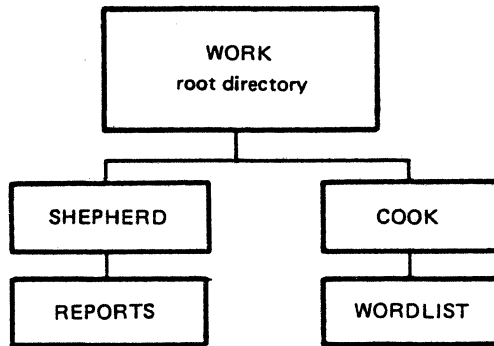
Figure 3-5. Location of Subordinate File REPORTS

Suppose you want to create a file named WORDLIST under the directory COOK. Since your working directory is still the root directory, WORK, enter the command:

**CR COOK>WORDLIST**

where COOK>WORDLIST is the relative pathname of the file you want to create. The file WORDLIST now resides immediately subordinate to the directory COOK, as shown in Figure 3-6.





86-051

Figure 3-6. Location of Subordinate File WORDLIST

### RENAMING FILES

Suppose that Cook wants to name his file more appropriately MATCHTM.D rather than MIME.D. The file can be renamed using the RN command:

```
RN MIME.D MATCHTM.D
```

### DELETING FILES

You can delete files using the Delete File (DL) command. For example, to delete the file DATAFILE in your working directory, enter:

```
DL DATAFILE
```

### COPYING FILES

The Copy command allows you to copy files between directories, into directories from a card reader, out of directories to a printer, and between tape or disk volumes. For example, suppose Cook wants to read cards into a file MATCHTM.D. From the home directory (COOK), enter:

```
COPY !CDR00 MATCHTM.D
```

All peripheral devices (tapes, card readers, and printers) are referred to by their symbolic peripheral device name; e.g., CDR for card reader.

When you read in cards, the card reader must be ready (the READY light must be on). While the command is being processed, your terminal locks; if the card reader is not ready or jams, the operator receives an error message. Until the card reader processes the end-of-file (EOF) card, the copy is not complete, and, if you fail to include an EOF card, the reader and your terminal remain locked. An EOF card is multipunched in column 1, rows 11, 5, 8, and 9. After copying is finished, you receive the Ready message or Master Menu (if in the Menu Subsystem).

Cook wants to copy an Assembly language subroutine, REC3.A, from his home directory, ^SYSVLA>UDD>PROGS>COOK, currently his working directory, to the directory ^SYSVLA>UDD>PROGS>TOOLS. Note the use of the relative pathname.

```
COPY REC3.A <TOOLS>REC3.A
```

The command allows you to omit the second argument if you are copying a file into your working directory. Thus, if Cook were in the directory ^SYSVLA>UDD>PROGS>TOOLS, and wanted to copy in the file ^SYSVLA>UDD>PROGS>COOK>REC3.A, he needs to type only:

```
COPY <COOK>REC3.A
```

The command copies REC3.A into TOOLS and names it REC3.A by default. You must be in the target directory to use this feature.

For another example, to copy cards onto a tape named BS001, that is already mounted, enter:

```
COPY !CDR00 !MT900>BS001>WESTNAMES
```

## LOCATING FILES

You can use the Where (WH) command to locate and display a file's full pathname. The system searches your working directory and the two system libraries, SYSLIB1 and SYSLIB2, looking for your file. If the file is found, its full pathname is displayed. If the file is not found, an error message is displayed. You may find this command useful if you know the simple pathname of a file but want to know its absolute pathname or to determine if the file you want to locate exists.

## LISTING FILES AND DIRECTORIES

You can list the contents of any directory that you have at least list access to by using the List Names (LS) command.

For example, Cook lists the contents of his working directory by entering:

```
LS -LG
```

```
DIRECTORY: ^SYSVL1>UDD>PROGS>COOK
```

| ENTRY NAME  | TYPE | PHYSICAL SECTORS | STARTING SECTOR HEX | RECORD LENGTH |
|-------------|------|------------------|---------------------|---------------|
| START_UP.EC | S    | 8                | 580                 | 256           |

\*\*\*\*\*  
\*\*\*\*\*

To determine the starting sector of the file for file dumping purposes, the record length is the number of characters per line. Listing Cook's file without any arguments would produce this information.

```
LS
```

```
DIRECTORY: ^SYSVL1>UDD>PROGS>COOK  
START_UP.EC S 8
```

```
TOTAL SECTORS 8
```

With no arguments, the LS command lists all files and directories subordinate to your working directory.

The -DIR argument of the command lists only directories subordinate to your working directory.

To stop output (scrolling) during execution of a list command, press the space bar on your terminal keyboard. To resume scrolling, enter @ on the same line, followed by a carriage return.

### Interrupting Execution

You can interrupt the execution of any command or program, at any time, by pressing the terminal break (BREAK) key. This signals the processor to interrupt execution. You can now enter any system command. To resume execution of the command or program, enter the Start (SR) command for programs or the Program Interrupt (PI) command for system software, such as the Editor or Linker.

If you do not want to resume execution after a break, you can use the Unwind (UW) command to return to command level, or a New Process (NEW\_PROC) command to restart your task group; i.e., return it to the state existing immediately after login.

For example, assume you are using the Line Editor to edit a large file, and you accidentally press the BREAK key while listing the file. To resume listing your file as if no break had occurred, you could issue an SR command, or you could save your work in the Editor and resume processing in edit mode by issuing a PI command. You can also enter the UW command if you want to close all your files, return to system command level, and proceed with other work. Your final option would be to issue a NEW\_PROC command that would return you to command level in your home directory.

### Controlling Output

Normally, all output goes to your terminal. (At login, for example, all four I/O files, user-in, user-out, command-in, and error-out, are assigned to your terminal.) If you are producing a large output, you may want to redirect it elsewhere. The following paragraphs describe how to direct your output to a file or to a printer.

#### DIRECTING OUTPUT TO A FILE

To direct output to a file (which need not have been previously created) using the FO (File Out) command, enter:

```
FO FILEA
```

All normal system output (such as a response to an LS command) goes to FILEA, which is your new user-out file. Error messages and the ready message that go to the error-out file cannot be redirected and continue to appear at your terminal. Thus, if you entered an LS command, the system writes the listing to FILEA and responds at your terminal with only the ready message. However, input directed to your terminal is unaffected by the FO command.

#### DIRECTING OUTPUT TO A PRINTER

If you are performing functions that lead to many pages of output, you can direct output to a printer. The command:

```
FO !LPT00
```

directs all subsequent output to LPT00 (assuming that you have access to the printer). Note that while you are using the printer, no one else can use it. In a multiuser system you may wish to avoid tying up the printer. (See "Deferred Printing" later in this section for information on printing large files.)

## REDIRECTING OUTPUT TO YOUR TERMINAL

After you have finished directing output to a printer, you should redirect output to your terminal. Enter the FO command with no arguments:

```
FO
```

(The default is to redirect output to your terminal.)

### Printing Control

You can print files at your terminal or you can request deferred printing. If you use the Print (PR) command, output appears on your terminal (i.e., output goes to the user-out file). This is inconvenient, however, if you are printing large files. For large files, you have the option of using deferred printing. The system stores your print request in a first-in, first-out queue.

## PRINTING FILES AT YOUR TERMINAL

If you want to print a file at your terminal, issue a PR command:

```
PR FILEB.D -SP
```

Remember that not all files are meant to be listed at a terminal. Some files are print files; some are not. Examples of print files are listings from the Linker and compilers and batch output files. In print files, the first character of each line is a print control character, instructing the printer how many spaces to skip between lines and when to skip to the top of a form. When printing a nonprint file, use the -SP argument of the Print command. This argument instructs the printer to print the first character of each line, and to skip one space between lines.

## DEFERRED PRINTING

To print large files, use the Deferred Print (DP) command. The DP command frees you from the need to reserve a printer and allows more efficient use of your system's printer resources. The request is queued on a first-in, first-out basis in one or more print queues.

Arguments of the DP command allow you to address your print output. If you are not at the printer, the person who is can separate printouts and route them to personnel. The Destination (-DS) argument accepts a string of up to 13 characters that appear as the first line of the address page. You can include blanks in the string if you enclose the string in quotes. The Header (HE) argument accepts a string of up to 26 characters that appear as the second line of the address page. For example:

```
DP START UP.EC -SP -HE "SHEPARD 692" -DS  
"WEST COMPUTER" -Q 3
```

produces the following printer output:

- An address page, with a header label and destination label
- One blank page
- One or more pages containing the file
- One blank page
- An end page, containing accounting information (e.g., the cost of the print job).

If you do not include a header and destination label, the default is the user name for the header label, and the account name for the address label.

Your request is automatically entered in the queue. In this example, Shepard's print request will not be executed until it reaches the head of print queue three.

#### NOTE

Deferred print requests are queued on disk and are not lost when the system is restarted.

#### Program Execution

Most of the programs you write require some type of input and output. Before you execute a program, you must provide information that tells the program where your input comes from and where your output will go. The GET and REMOVE commands allow you to reserve files and devices for program input and output, and, after program execution, to cancel those reservations.

GET performs two functions. First, it reserves a file or device for use by the executing program. This reservation may set exclusive access or some degree of shared access (see "Reserving Files or Devices"). Secondly, GET establishes a relationship between pathnames and the logical file numbers (LFNs) by which you can gain access to files and devices. Using a GET command overrides any internal LFN assignments you have included in your FORTRAN, BASIC, or Assembly language program.

Once program execution has terminated, you can use the REMOVE command to cancel file/device reservations and the LFNs that your program assigned with the GET command.

For example, if you are compiling the COBOL program CARDIN, CARDIN uses two files, a card reader (from which input will be read), and a disk file (to which output will go). The program refers to these two files by internal file names (IFNs) 0A and 0C, which correspond (map) to logical file numbers (LFNs) 1 and 3. You assigned these IFNs when you wrote your COBOL program. The system maps these IFNs to corresponding LFNs.

After linking your object unit into a bound unit, you must use the GET command to reserve an input file (a card reader) and an output file (a disk file).

To reserve the card reader, specify:

```
GET !CDR00 -LFN 1
```

To reserve the disk file, specify:

```
GET COBOL_DIR>MASTER -LFN 3
```

In this example, it is assumed that the file MASTER was previously created under the directory COBOL\_DIR. It is also assumed that the directory COBOL\_DIR is subordinate to your working directory. (The GET command could have directed program output to any file, not necessarily one named MASTER.)

If you have already loaded the card reader, you can now execute CARDIN by entering the simple pathname (since the bound unit CARDIN is in your working directory):

```
CARDIN
```

The program reads cards into the file MASTER. Once the program terminates, remove the device and file reservations:

```
REMOVE !CDR00 -LFN 1
```

```
RDY:
```

```
REMOVE COBOL_DIR>MASTER -LFN 3
```

```
RDY:
```

## Reserving Files or Devices

You can use the GET command to reserve a file or device for use by your task group. When you reserve a file, you can specify whether other users will be allowed some form of concurrent access.

For example, when you reserve a disk file, you can specify that all users can read the file while you have it reserved, but that only you can alter (write to) it. To do this, enter:

```
GET FILEB -LFN 1 -SHARE R
```

If a directory is reserved exclusively for you (using the -SHARE N argument), then all subdirectories and files are also reserved exclusively. Thus, entering:

```
GET ^VOL04 -SHARE N
```

reserves the entire volume VOL04 for your exclusive use. Note that the system always reserves tapes exclusively for your use when you reserve them.

## Communicating With Other Users

You can send messages to the system operator and send (or receive) mail to other system users by using the Message (MSG) or MAIL commands. Messages sent to the operator are displayed immediately on the operator terminal; mail is not displayed until the receiver enters the MAIL command.

To send a message to the operator, you might enter the following request:

```
MSG "PLEASE MOUNT ^VOLA"
```

You must enclose your message in quotation marks (or apostrophes) if it contains embedded blanks.

To send mail to another person, you might enter:

```
MAIL LOWELL
```

where LOWELL is the person\_id of the receiver. The system responds:

INPUT:

You can then enter the text of your message. Terminate the message by entering a period (.) or the letter Q followed by a carriage return. Your message is queued in Lowell's mailbox until Lowell issues a MAIL command to display mail.



To mail a file that might be a program or a long message for many users, use the filename argument of the MAIL command:

```
MAIL LOWELL HEX_AS.A
```

This command mails the file HEX\_AS.A to Lowell. Long messages should not be sent to users at a VIP terminal.

#### NOTE

Before you can receive mail, either you or your system operator must have previously created the mailbox directory and the necessary mailboxes, and have set access controls on these mailboxes. See the System User's Guide for details.



REMOVE THIS PAGE AND PLACE TAB FOR

TAB 4

SCREEN EDITOR



## **Section 4**

# **SCREEN EDITOR**

The Screen Editor is called Screen-Oriented Program for Editing Operations (SCORPEO). It is a full screen, interactive text editing, and documentation preparation tool for GCOS 6 ASCII files. SCORPEO uses the entire screen of a visual display terminal to view and manipulate data, thereby making text editing faster and simpler.

This section describes the Screen Editor capabilities as well as the directives used to create, modify, and save files.

### OVERVIEW

The Screen Editor creates and/or alters character text in a file. You control editing by using a combination of directives, function keys, and labeled keys.

Commonly used editing operations are assigned to function keys. To view a data file, use selected function keys to move a full screen of text (window) up and down within the file. To view lines that exceed the width of the terminal, move the window left or right. Other function keys move the cursor in a forward or backward movement in units of words. At all times the cursor positioning keys (arrow keys) can be used to place the cursor in any location within the textual window.

Labeled keys perform the function described on the key. To edit text quickly, simply position the cursor over any character(s) and overstrike (or overtype) them. Lines and/or characters can be inserted or deleted relative to the cursor position.

SCORPEO also provides global file operations, such as positioning, searching, and changing operations. In addition, line numbers can be used to rapidly fill the window with a view of specific lines within the file. Where applicable, these global operations are compatible with similar capabilities in the GCOS 6 Line Editor (ED). You can edit files created by the Line Editor with the Screen Editor. The Line Editor is described in Section 5.

Tab stops for COBOL, FORTRAN, and GCOS 6 Assembly language statement formats are built into SCORPEO. For details on developing source programs, see the appropriate language manual.

All editing is done in a temporary work area called a buffer. This buffer references a pair of temporary work files. When you invoke SCORPEO, a buffer and associated work files are automatically created for you. To save the Screen Editor output, you must write the contents of the buffer to a file.

#### NOTE

During a single execution of the Screen Editor, you can read only one file. You must write the contents of the buffer out to another file or to the same file. To edit a second file, you must reinvoke the Screen Editor. (See "Loading the Screen Editor" later in this section.)

#### SCREEN EDITOR PROCESSING

You control Screen Editor processing by specifying directives, or using function keys or labeled keys. The subsections that follow describe the following Screen Editor processing functions.

- Suffix Conventions--describes file naming conventions.
- Loading the Screen Editor--describes the command used to invoke the Screen Editor.
- Description of the Screen--defines and illustrates the three regions of the screen.
- Creating a Source Unit--describes the procedure for creating a file.
- Changing an Existing Source Unit--describes the procedure for modifying a file.

- Interrupting Screen Editor Processing--describes the procedures used to stop Screen Editor processing.
- Entering Screen Editor Directives--defines the rules for entering Screen Editor directives.
- Screen Editor Directive Format Conventions--defines the rules for specifying directive formats.
- Designating Lines--describes the procedure used for locating, adding, and deleting lines from a file.
- Special Characters--defines the characters used to specify a processing function.
- Screen Editor Directives--defines each directive in alphabetical order by directive name. Provides the information necessary for using the directives to create and modify files.
- Function Keys--defines each function key in alphabetical order by function name. Describes function key use in creating and modifying files.
- Labeled Keys--describes each labeled key as it is used by the Screen Editor in creating and modifying files.

#### SCREEN EDITOR SUFFIX CONVENTIONS

When you create a source unit, you should append the appropriate suffix identification character to the name of the file that will contain the source unit. The suffix designates the type of programming language that constitutes the source unit. The suffix must be .C for COBOL or C programs, .F for FORTRAN programs, .B for BASIC programs, .PS for Pascal programs, .AS for ADA programs, and .A or .P for Assembly language programs.

When you specify the file names of Screen Editor input and output files (when calling the Screen Editor, and in selected directives), you must designate the complete file name, including the suffix that denotes the contents of the file. The Screen Editor does not append a suffix to its input and output files.

#### LOADING THE SCREEN EDITOR

To load the Screen Editor when running under the menu subsystem, select the SCORPEO option from one of the selection menus that contains it, and press TRANSMIT. The Screen Editor can be loaded when you enter the menu subsystem by selecting Commonly Used Functions (CF) from the General Menu System menu and then selecting SCORPEO (SC) from the Commonly Used Functions menu.

To load the Screen Editor by a command line, enter the SCORPEO command.

FORMAT:

```
SCORPEO [path] [ { -LINES } nnnn ]
                { -L }
```

ARGUMENTS:

None or any number of the following control arguments may be entered:

[path]

Pathname of the file you wish to edit. Pathnames can be full or relative. If a pathname is relative, it is expanded relative to your current working directory. If you are creating a new file without any initial input from another file, do not enter a pathname. If you specify path, the first 17 lines of the file will fill the window (text region).

```
{ -LINES } nnnn
{ -L }
```

Approximate number of lines the Screen Editor should hold in main memory during the current editing session. nnnn is a positive decimal value between 500 (the minimum number of lines you can declare) and 4096 (the maximum). If you specify less than 500 lines or more than 4096, an error is reported. When editing very large files (over 10,000 lines) the -Lines option can help improve performance. A value higher than the 1000 line default (from 2000 to 4096) is recommended. For files over 35,000 lines, use a value of at least 1500. The -Lines option increases memory usage (-LINES nnn \* 22 words) for internal structures.

Default: 1000 lines.

Once you have loaded the Screen Editor, a screen such as that described in Figure 4-1 or Figure 4-2 is displayed. If you are creating a file (you invoked the Screen Editor without a pathname), the screen shown in Figure 4-1 is displayed. If you are modifying (editing) a previously created file, a screen similar to that shown in Figure 4-2 is displayed.

If you are editing a file that contains characters from the 8-bit ASCII Extended Character Set (80 through FF hexadecimal), be sure your terminal supports these characters (i.e., HDS 2 terminal configured for 8-bit operation).



Description of the Screen

The display on the terminal used by the Screen Editor is broken down into three distinct areas called "regions." Each region is described in detail in the following paragraphs. Refer to Figure 4-1 and Figure 4-2 for the location of each region.

```
LEFT MARGIN = 001      CURRENT LINE = 00001
*****TOP OF FILE***** TOP OF FILE *****
```

(17 blank lines)

-----  
DIRECTIVE:

Figure 4-1. Sample Screen for Creating a File

```
^VOL1>DIR>INVTNRY      MODIFIED
LEFT MARGIN = 001      CURRENT LINE = 00001
*****TOP OF FILE***** TOP OF FILE *****
```

(17 lines of text are displayed here)

-----  
DIRECTIVE:

Figure 4-2. Sample Screen for Modifying a File

## STATUS REGION

The status region of the screen is that area at the top of the screen that shows the status of the file.

When you are creating a file, the information displayed in the status region is: the current position of the left margin and the current line number (as shown in Figure 4-1).

When you are editing a file, the information displayed is:

- The full pathname of the file you are editing
- The flag MODIFIED (if modified)
- The current position of the left margin (as shown in Figure 4-2).

## TEXT REGION

The text region of the screen is the large area in the middle of the display where you actually create and modify the text. The text region is also called the "window." The window is 18 lines long and from 80 to 256 characters wide. (You control the width of the text with the Window Width directive described later in this section.) The maximum record length of a file is 256 characters.

When the first 17 lines of a file appear in the window, you will see at the top of the window a line designated as TOP OF FILE. This line is called the control line. You can use this line to verify that you are at the beginning of the file and to position the cursor when you want to insert text before the first line of your file. This line does not appear within your file.

At the bottom of the window is a line of dashes and vertical bars. This line is called the tab designator line. The position of the vertical bars indicates the current tab stops. To set these tab stops, use the Language Type directive or the TAB SET key. To clear these tab stops, use the TAB CLR or the CLR/TAB/SET keys. Both operations are described later in this section.

## DIRECTIVE REGION

The directive region is at the bottom of the screen just under the tab stop designator line. The cursor is positioned here when you press the HOME key to enter directives to the Screen Editor. Screen Editor directives are described later in this section.

Immediately below the directive region is the area where you can view system messages, both informational and diagnostic, that relate to the current editing activity.

## Creating a Source Unit

To create a source unit, perform the steps that follow. Directives and function keys are described later in this section.

1. Change the working directory to a user volume by specifying the Change Working Directory command (see the Commands manual) or by using the CWD form.
2. Call the Screen Editor (see "Loading the Screen Editor" earlier in this section).
3. Enter the source unit text.
4. Make changes, if necessary, by entering the appropriate directives, by using the function keys or the labeled keys, or by typing over existing text.
5. Write the contents of the buffer to a file by using the Write directive.
6. Exit from the Screen Editor by entering the Quit directive.

You can use tab stops when creating a file (that is, with the TAB SET labeled key or LANGUAGE TYPE directive). Tab characters convert to the appropriate number of space characters according to the tab stops, and are not present in the file. When an existing file that contains tab characters is modified, the tab characters are converted to spaces.

## Changing an Existing Source Unit

When you read an existing file, any ASCII non-printable characters are converted to dots (hexadecimal value 2E). If the file contains any 8-bit ASCII extended characters (hexadecimal 80 through FF) and your terminal doesn't support 8-bit operation, the 8th bit is folded and the character is displayed as a 7-bit character. To change an existing source unit, perform the steps that follow. (You can change a source unit that was created using the Line Editor with the Screen Editor.) Directives and function keys are described later in this section.

1. Change the working directory to a user volume by specifying the Change Working Directory command (see the Commands manual) or by using the CWD form.
2. Call the Screen Editor (see "Loading the Screen Editor" earlier in this section) optionally specifying the pathname of the source file you wish to modify.
3. If you did not specify a file pathname when you called the Screen Editor, use the Read directive to read into the buffer the source unit you wish to edit.

4. Use the appropriate Screen Editor directives, function keys, and labeled keys or simply type over existing text to modify the source unit.
5. Write the contents of the buffer to the file from which the lines were read or to a different file by using the Write directive.
6. Exit from the Screen Editor using the Quit directive.

### Interrupting Screen Editor Processing

You can interrupt Screen Editor processing by either:

- Entering the Quit directive.

Entering the Quit directive is the preferred way to terminate processing. If no modified buffer exists (i.e., the user has not changed a line in the file, or has not written a newly created file), then the Screen Editor terminates when the Quit directive is entered. If a modified buffer exists, then a question "Modified buffer exists. Do you still wish to quit? (Answer yes or no.)" is displayed and you make a choice (yes = terminate; no = resume). Thus, the system does not process the Quit directive, but the Screen Editor does in a very specific way, as described here.

- Pressing the INTERRUPT or BREAK key on your terminal.
- Entering  $\Delta$ CABgroup-id on the operator terminal, where group-id is the two-character task group name of the group containing the Screen Editor task.

A **\*\*BREAK\*\*** message is displayed on the 25th line of your terminal when the system interrupts the Screen Editor. Before the break message can be responded to, it must be acknowledged by pressing the TRANSMIT key. Pressing TRANSMIT clears line 25 of your terminal. At this point, there are four actions you can take:

1. Enter any user command found in the Commands manual.
2. Enter the Start (SR) command. The Screen Editor resumes processing as if it had not been interrupted. All of the changes to the file you were editing at the time of the interrupt are intact.
3. Enter the Unwind (UW) command. The Screen Editor terminates processing and the system returns to the command level. None of the changes made to the file since the last write directive are saved.

Using UW causes the Screen Editor to terminate unconditionally.

4. Enter the New Process (NEW PROC) command. The current task group is aborted and then restarted using the same arguments specified when you logged in. None of the changes you made to the file since the last write directive are saved.

#### ENTERING SCREEN EDITOR DIRECTIVES

To enter any of the Screen Editor directives, press the HOME key to position the cursor in the directive region of the screen. See the description of the HOME key later in this section under "Labeled Keys." After entering the directive and its arguments (if any), press the RETURN key to execute the directive.

If you have entered a directive line and you decide not to use the directive before you press the RETURN key, you can cancel the directive by pressing the Cursor Up (↑) key. (This key is described in detail later in this section.) The cursor is placed in the text region at the location on which the cursor was positioned before you pressed the HOME key.

#### Screen Editor Directive Format Conventions

Most Screen Editor directives consist of only a directive name, a directive name preceded by one or two line numbers, a directive name optionally preceded by one or two line numbers and followed by text, or only a line number. You cannot specify a directive line longer than 70 characters. You can specify only one directive on a line. Directive formats are:

FORMAT 1:

line\_number

FORMAT 2:

dirname

FORMAT 3:

line\_number    dirname

FORMAT 4:

line\_number ,line\_number    dirname

FORMAT 5:

dirname    text

FORMAT 6:

line\_number    dirname    text

## FORMAT 7:

line\_number ,line\_number dirname text

### NOTES

1. If a directive includes text, you must leave at least one space between the directive name and the text.
2. If you specify two line numbers in a directive line, separate them by a comma (,); do not use any spaces.

### DESIGNATING LINES

You can locate each line in the buffer by entering a decimal number that indicates the file-relative position of the line within the buffer. The first line in the buffer is line 1; subsequent lines are numbered sequentially in ascending order.

Screen Editor directives can cause lines to be added to or deleted from the buffer. Each time this occurs, all succeeding lines are renumbered. For example, if line 5 is deleted, line 6 becomes 5, and each subsequent line number is decremented by 1.

If you specify a line that is not in the buffer, an error message is displayed.

### BLOCK DESCRIPTION

SCORPEO provides a set of block-related functions to move collections of data within a file and to extract collections of data from a file. The Block function key defines a "block," or subset of the buffer on which some action will be performed. Function keys are used to define blocks, and subsequently to move, copy, or delete them. Blocks can also be written to external files.

You define a block by positioning the cursor on the desired starting position and pressing the Block function key. Next, you position the cursor on the ending position and again press the Block function key. It is not necessary to set the starting position of the block first. The starting block position is always considered to be the block definition closer to the beginning of the buffer. The ending block position is always considered to be the block definition closer to the end (last line) of the buffer.

A block is defined by its location; i.e., the line and column numbers of its starting and ending points. For example, if you define a block beginning in line 1, column 1, and ending in line 10, column 80, and then you delete lines 5 through 10, the resulting block begins at line 1, column 1, and ends at the old line 16 (now the new line 10).

Once you define a block, it can be acted upon. You perform actions on a block using the Move Block, Copy Block, Erase Block, and Delete Block function keys, and the Write Block and Change Block directives.

The following block directives and block-related function keys erase the definition of the block:

- Delete Block
- Erase Block
- Move Block.

The following block directives and block-related function keys preserve the definition of the block:

- Lower Case
- Upper Case
- Change Block
- Write Block
- Copy Block.

These function keys and directives are described later in this section.

When defining a block for a Move Block or Copy Block directive, it is possible to split a line. A line split occurs when you try to insert or delete a block other than at the end points of a line. If you do this, the lines are truncated (if you insert) or concatenated (if you delete). Spaces are considered trivial characters and are truncated without a warning message. If concatenation causes an overflow of the maximum line length, the overflowing characters are truncated.

The Delete Block function key erases the block definition and deletes all the text in the block. If both the block start and block end positions are split lines, the two split lines at the end of each block are concatenated. The result is the display of one line where the block definition had been previously. For example:

The following lines are in the window with the block start and end positions denoted by shaded rectangles.

```
THIS IS AN EXAMPLE TO SHOW WHAT HAPPENS  
WHEN THE TEXT IS DELETED BY USING  
THE DELETE BLOCK FUNCTION KEY WITH SPLIT LINES.
```

Pressing the Delete Block function key results in the following:

```
THIS IS AN EXAMPLE WITH SPLIT LINES.
```

Use of the Copy Block function key has a possibility of two different split lines. The left portion of the line to which you are copying (up to the cursor position) is concatenated to the block start position. The right portion of the line on which you are copying is concatenated to the block end position. The result is the same as if you inserted characters. For example:

The following lines are in the window with the block start and end positions designated by shaded rectangles. The position at which you want to copy is designated by an arrow.

THIS IS AN EXAMPLE OF COPY BLOCK FUNCTIONALITY.  
THE DEFINED BLOCK WILL BE COPIED AT THE CURSOR POSITION.  
THIS LINE IS THE START OF THE COPY BLOCK.  
ALL THE TEXT IN THE BLOCK WILL BE COPIED  
TO THE COPY POSITION. THIS SHOWS THAT THE BLOCK WILL NOT BE  
DELETED.

Pressing the Copy Block function key produces the following:

THIS IS AN EXAMPLE OF THE COPY BLOCK.  
ALL THE TEXT IN THE BLOCK WILL BE COPIED  
TO THE COPY POSITION. THIS SHOWS COPY BLOCK FUNCTIONALITY.  
THE DEFINED BLOCK WILL BE COPIED AT THE CURSOR POSITION.  
THIS LINE IS THE START OF THE COPY BLOCK.  
ALL THE TEXT IN THE BLOCK WILL BE COPIED  
TO THE COPY POSITION. THIS SHOWS THAT THE BLOCK WILL NOT BE  
DELETED.

The Move Block function key operates the same with split lines as does the Copy Block with a Delete Block. The split lines are the same as shown previously, except that the block is deleted from its previous position and the left portion of the block start line is concatenated to the right position of the block end line. Using the same example as used in the Copy Block example above, pressing the Move Block function key results in the following:

THIS IS AN EXAMPLE OF THE COPY BLOCK.  
ALL THE TEXT IN THE BLOCK WILL BE COPIED  
TO THE COPY POSITION. THIS SHOWS COPY BLOCK FUNCTIONALITY.  
THE DEFINED BLOCK WILL BE COPIED AT THE CURSOR POSITION.  
THIS LINE IS THE START OF THAT THE BLOCK WILL NOT BE  
DELETED.

The Write Block directive writes a split line as a line by itself. For example assume you defined a block as shown:

MOVE PAY TO OUT\_PAY.  
MOVE CREDIT\_UNION TO OUTPUT\_CU.  
MOVE FED\_TAXES TO OUTPUT\_FED\_TAX.



Using the Write Block directive results in the following:

```
TO OUT_PAY.  
MOVE CREDIT_UNION TO OUTPUT_CU.  
MOVE FED_TAXES
```

The Change Block directive is not affected by split lines.

## SPECIAL CHARACTERS

When the following ASCII characters are included in search expressions or change expressions, they have special meanings. All special characters can be used only in search expressions, except the ampersand (&) special character, which can only be used in change expressions.

| <u>Character</u> | <u>Description</u>  |
|------------------|---|
| *                | Requests expressions that contain any number (or none) of the preceding character(s). If this character is the first character of a regular expression, it has no special meaning.  |
| ^                | When designated as the <u>first</u> character of an expression, requests <u>lines</u> that begin with the specified expression (excluding the character ^).                         |
| \$               | When specified as the <u>last</u> character of an expression, requests <u>lines</u> that end with the specified expression (excluding the character \$).                            |
| .                | Can be any character on a line; specify one per character (e.g., ".." means any two characters on any line).  |
| &                | Can be used only in the change expression of a change directive to indicate that the strings of characters following "&" are to be concatenated to the target string of the search. |
| !C               | Requests that the following character not be interpreted as a special character (e.g., "!C*" means match an asterisk). Specify the C in uppercase.                                  |
| [n]x             | Requests a repeat factor ([n]) of the specified character (x). x can be any character including a period. (e.g., "[25]." matches any 25 columns of characters).                     |

## Summary of Screen Editor Directives

Table 4-1 lists each Screen Editor directive mnemonic, summarizes its function, and designates the directive name under which it is more fully described.

Table 4-1. Summary of Screen Editor Directives

| Directive Mnemonic | Function  | Directive Name |
|--------------------|---|----------------|
| BL                 | Display last line of buffer.  | Bottom Line    |
| C                  | Change one character string to another character string.                                | Change         |
| CA                 | Change all occurrences of a character string in the buffer to another character string. | Change All     |
| CB                 | Change all occurrences of a character string in a block to another character string.    | Change Block   |
| line number        | Display a line of text.   | Display        |
| LC                 | Convert all uppercase characters in a block to lowercase.                               | Lower Case     |
| LM                 | Display the left margin of the buffer.  | Left Margin    |
| LT                 | Set tab stops for the specified programming language.                                   | Language Type  |
| Q                  | Conditionally terminate execution of the Screen Editor.                                 | Quit           |
| R                  | Read text from the specified file to the buffer.  | Read           |
| RM                 | Display the right margin of the buffer.   | Right Margin   |
| S                  | Search the buffer for the specified character string.                                   | Search         |

Table 4-1 (cont). Summary of Screen Editor Directives

| Directive Mnemonic | Function  | Directive Name  |
|--------------------|---|-----------------|
| SB                 | Search the buffer for the specified character string from the current cursor position backward to line 1.                     | Search Backward |
| SC                 | Change the number of lines to scroll.   | Scroll Change   |
| SF                 | Search the buffer for the specified character string from the current cursor position forward to the last line of the buffer. | Search Forward  |
| TB                 | Do not suppress trailing blanks when text is written to a file.   | Trailing Blanks |
| TL                 | Display line 1 of the buffer.   | Top Line        |
| UC                 | Convert all lowercase characters in a block to uppercase.   | Upper Case      |
| V                  | Display the current version of the Screen Editor.   | Version         |
| W                  | Write the contents of the buffer to a file.   | Write           |
| WB                 | Write the specified block of text in the buffer to a file.  | Write Block     |
| WW                 | Set the window width to the specified value.  | Window Width    |

SCREEN EDITOR DIRECTIVES

Screen Editor directives are described in detail on the following pages. In the examples, numbers in parentheses are references to line numbers and do not appear in memory or in the text.

# BOTTOM\_LINE

## BOTTOM LINE (BOTTOM LINE OR BL)

Display the last line of the buffer at the top of the current window.

The cursor is positioned on the last line (the bottom line) of the file in the same column in which it was positioned before it was moved to the directive line.

FORMAT:

{ BOTTOM\_LINE }  
{ BL }

Example:

BL

Display the last line of the buffer at the top of the current window.

# CHANGE

## CHANGE (CHANGE OR C)

Search the buffer for the specified search expression and replace the first occurrence of the search expression with the change expression.

Searching begins at the current cursor position. Searching proceeds in a forward direction until the end of the file is reached, and, if necessary, resumes at the top of the file and proceeds forward to the current cursor position.

The search expression must be found wholly on a line of the file to be considered a matched string.

When the directive has completed execution and a match was found, the cursor rests on the first character of the changed expression. The changed line is displayed as the first line in the window.

If a match was not found, the message SEARCH FAILED is displayed.

It is not necessary to repeat the search expression for subsequent identical changes. Simply entering the first two delimiters and the change expression changes the next occurrence of the search expression to the change expression.

### FORMAT:

```
{ CHANGE } "search_expression"change_expression"  
{ C }
```

### ARGUMENTS:

"

(Delimiter) Can be any character. You must use the same character in each of the three locations where a delimiter is required. If using a delimiter that is a character within the search expression of the change expression, you must use the special character !C before the character within the text. It is recommended that you use a delimiter that is not within the search expression or the change expression.

## CHANGE

### search\_expression

Character string for which the Screen Editor is searching. The first occurrence of this character string is replaced with the character string specified in the change expression.

### change\_expression

Character string that replaces the first occurrence of the argument search expression.

#### Example 1:

```
C "ABC"DEF"
```

Change the next occurrence in the file of ABC to DEF.

#### Example 2:

```
C ""DEF"
```

Change the next occurrence of the previously defined search expression ABC to DEF.

#### Example 3:

```
C "^.*$"DEF"
```

Change the next line to DEF.

#### Example 4:

```
C ""
```

Delete the next occurrence of the previously designated search expression.

## CHANGE\_ALL

### CHANGE ALL (CHANGE ALL OR CA)

Search the buffer for all occurrences of the specified character string and replace all occurrences of the character string with another specified character string.

Each occurrence of the search expression must be found wholly on a line of the file to be considered a match string.

After this directive is executed, the cursor is positioned on the first character of the last changed character string. The changed line is displayed as the first line of the window.

#### FORMAT:

[n[,m]] { CHANGE\_ALL } "search\_expression"change\_expression"  
          { CA }

#### ARGUMENTS:

[n[,m]]

Includes the starting line number (n) and ending line number (m) in which the specified search expression is changed. If you specify both starting and ending line numbers, all occurrences of the specified search expression found between the line numbers are changed. If you specify only a starting line number, only those occurrences of the search expression from the specified line number to the end of the buffer are changed. If you do not specify line numbers, the search for the search expression begins at line 1. Searching proceeds in a forward direction until the end of the file is reached and resumes at the top of the file until all occurrences of the search expression are replaced. When the change is completed, the cursor rests on the last changed expression.

"

(Delimiter) Can be any character. You must use the same character in the three locations where a delimiter is required. If using a delimiter that is a character within the search expression or change expression, you must use the special character !C before the character within the text. It is recommended that you use a delimiter that is not within the search expression or the change expression.

## CHANGE\_ALL

### search\_expression

Character string for which the Screen Editor is searching. Each occurrence of this string according to the line number values specified by n or m (see above) is replaced with the character string specified in the change expression argument.

### change\_expression

Character string that replaces each occurrence of the search expression.

#### Example 1:

```
CA "ABC"DEF"
```

Change all occurrences in the buffer of ABC to DEF.

#### Example 2:

```
5,10 CA "ABC"DEF"
```

Between (and including) lines 5 to 10, change all occurrences of ABC to DEF.

#### Example 3:

```
5 CA "ABC"DEF"
```

Between (and including) line 5 and the last line of the buffer, change all occurrences of ABC to DEF.



# CHANGE\_BLOCK

## CHANGE BLOCK (CHANGE BLOCK OR CB)

Search the current block for all occurrences of the specified expression and replace each occurrence of the expression with another specified expression.

Before using this directive, you must define the block of text you wish to change (see the description of the Block function key under "Function Keys" later in this section).

See "Block Description" earlier in this section for details on blocks.

### FORMAT:

```
{ CHANGE_BLOCK } "search_expression"change_expression"  
CB
```

### ARGUMENTS

"

(Delimiter) Can be any character. You must use the same character in the three locations where a delimiter is required. If using a delimiter that is a character within the search expression or change expression, you must use the special character !C before the character within the text. It is recommended that you use a delimiter that is not within the search expression or the change expression.

### search\_expression

Character string for which the Screen Editor is searching. Each occurrence of this string within the defined block is replaced with the character string specified in the change expression argument.

### change\_expression

Character string that replaces each occurrence of the search expression.

## CHANGE\_BLOCK

Example:

You have previously defined a block as

```
C:=A-B,D:=C-B,E:D-C,
```

Enter the directive

```
CB  ",",""
```

When the directive is executed, the resulting block is:

```
C:=A-B;D:=C-B;E:=D-C;
```

# DISPLAY

## DISPLAY

Display a specified line of text.

After executing the Display directive, a new page (window) of text is displayed. The specified line of text appears as the first line of the new window.

The cursor is positioned on the new line in the same column in which it was positioned before you executed the directive.

### FORMAT:

line\_number

### ARGUMENT

line\_number

Line number (decimal) of the text you wish displayed. The line number must be a positive integer whose maximum value is 65535. The specified line appears at the top of the window.

### Example:

35

Display line 35 of the buffer on the first line of the window.

# LANGUAGE\_TYPE

## LANGUAGE TYPE (LANGUAGE TYPE OR LT)

Set tabs stops for the specified programming language.

FORMAT:

$\left. \begin{array}{l} \text{LANGUAGE\_TYPE} \\ \text{LT} \end{array} \right\} [\text{language}]$

ARGUMENT:

language

Programming language in which you are creating or editing your source file. Specify the language as shown below to set the appropriate tab stops:

| <u>Language</u> | <u>Tab Stops (Column)</u>                                  |
|-----------------|--|
| (default)       | 11, then every 10 columns                                  |
| Assembly or A   | 11, then every 10 columns                                  |
| COBOL or C      | 8, 12, 21, then every 10 columns through column 61, and 73 |
| FORTTRAN or F   | 7, 11, 21, then every 10 columns through column 61, and 73 |
| PASCAL or P     | 11, then every 10 columns                                  |
| BASIC or B      | 11, then every 10 columns                                  |

If you do not specify "language," the tab stops are set as specified in the default tab stops above.

Example:

LT COBOL

Set tab stops at columns 8, 12, 21, and then every 10 columns until 61; then the next tab stop is 73. The tab stop line at the bottom of the text region is changed accordingly.

## LEFT\_MARGIN

### LEFT MARGIN (LEFT MARGIN OR LM)

Display the left margin of the buffer in the current window.

The left margin is always set to the first character of each line.

The cursor is positioned in column 1 on the line on which it was positioned before you executed the directive.

See the Window Left function key description later in this section.

FORMAT:

{ LEFT\_MARGIN }  
{ LM }

Example:

LM

Display the first 80 columns of text in the current window.

# LOWER\_CASE

## LOWER CASE (LOWER CASE OR LC)

Convert all uppercase characters in a previously defined block to lowercase characters.

If the block contains characters within apostrophes (') or quotation marks ("), these characters are not converted.

You must have previously defined a block before you can use this directive. See the Block function key description later in this section for information on defining blocks.

FORMAT:

{ LOWER\_CASE }  
{ LC }

Example:

Assume you have defined the following block:

THIS PROGRAM CALCULATES THE WEEKLY 'GROSS' AND 'NET' PAY

Enter the Lower Case directive:

LC

The block now reads:

this program calculates the weekly 'GROSS' and 'NET' pay

# QUIT

## QUIT (QUIT OR Q)

Terminate the current screen editing session and close the file associated with it.

You must specify the Quit directive at the end of a screen editing session.

Quit is executed conditionally. If you have modified a file and enter the Quit directive without having saved (written) the file (see the Write directive later in this section), you are warned that a modified file exists. If you want to save the edited text, answer NO or N to the prompt "Modified buffers exist. Do you wish to quit? (Answer yes or no.)," and enter a Write directive to save the file. Now enter the Quit directive. If you do not wish to save the modified text, answer YES or Y to the prompt.

### NOTE

If the cursor is positioned past the last visible character on a line, blanks exist on the line up to the cursor. If a write is performed with the cursor positioned this way, these blanks are suppressed during the write operation (unless the TB option is used). The cursor returns to the column past the last visible character on the line rather than to its original position on the screen. If the TB option is used and the cursor is positioned past the last visible character on the line, the cursor returns to its original position on the screen after the write operation and trailing blanks remain on the line.

The Quit function key operates exactly as the Quit directive.

FORMAT:

$$\left. \begin{array}{l} \text{QUIT} \\ \text{Q} \end{array} \right\}$$

Example:

Q

Terminate the current screen editing session.

# READ

## READ (READ OR R)

Place the contents of the specified file into the buffer.

The cursor is positioned in column 1 of the first line of the file.

If you have not specified the pathname of the file when you called the Screen Editor (see "Loading the Screen Editor" earlier in this section), use this directive first to read in the file you wish to edit.

You may only use the Read directive once during the current screen editing session (i.e., you may only edit one file at a time).

File concurrency for the specified file is exclusive read and exclusive write.

During the read, all tab characters are replaced with the appropriate number of blanks according to the currently defined tab stops. If this occurs, the MODIFIED status flag is displayed. When the file is saved (written), it will contain no tab characters.

During the read, if any hexadecimal sequence contains an ASCII non-printable character (i.e., hexadecimal characters 00 to 1F and 7F to 9F), each ASCII non-printable character is replaced by the ASCII period (.) character, and the MODIFIED status flag is displayed. During the read on a 7-bit terminal, if any hexadecimal sequence contains an extended 8-bit ASCII character in the range of A0 to FE, each character is folded and displayed as a 7-bit character.

### FORMAT:

$\left. \begin{array}{l} \text{READ} \\ \text{R} \end{array} \right\} \text{ path}$

### ARGUMENT:

path

Pathname of the file to be read. Can be any valid form of pathname.

### Example:

R FILEA

Read the contents of the file FILEA into the buffer.



## RIGHT\_MARGIN

### RIGHT MARGIN (RIGHT MARGIN OR RM)

Display the right margin of the buffer in the current buffer.

The current window is moved to the right so that column 80 of the display coincides with the column that is the current window width.

The cursor is positioned in the last column of the line on which it was positioned before you executed the directive.

Use this directive to view text beyond column 80.

See the Window Right function key described later in this section.

FORMAT:

{ RIGHT\_MARGIN }  
{ RM }

Example:

RM

Display the right margin of the buffer in the current window.

# SCROLL\_CHANGE

## SCROLL CHANGE (SCROLL CHANGE OR SC)

Change the number of lines that move through the text region (window) when you press the Window Up or Window Down function keys. (The Window Up and Window Down function keys are described under "Function Keys" later in this section.)

### FORMAT:

$\left. \begin{array}{l} \text{SCROLL\_CHANGE} \\ \text{SC} \end{array} \right\} \quad [\text{lines}]$

### ARGUMENT:

[lines]

Number of lines to move the current window. Can be any positive decimal integer. If a boundary (top or bottom line) occurs before the specified number of lines are scrolled, the boundary is displayed and scrolling stops. This value remains in effect until explicitly changed by another Scroll Change directive.

Default: 18 lines.

### Example:

If the current window displays the lines

(1)  
(2)  
(3)  
(4)  
(5)  
.  
.  
.  
(18)

and you enter the directive line SC 4 and press the Window Down function key, the current window will display:

(5)  
(6)  
(7)  
.  
.  
.  
(22)

# SEARCH

## SEARCH (SEARCH OR S)

Search the buffer for the specified search expression (character string).

The cursor is positioned on the first character of the matched search expression. The line containing the match is displayed on the first line of the window.

If you have previously defined a search expression, simply entering the directive followed by the two identical delimiters searches for the next occurrence of the search expression.

Searching begins at the current cursor position, continues to the end of the file, and, if no match is found, begins at the top of the buffer (line 1 of the file) and continues to the current cursor position.

If no match is found, the message SEARCH FAILED is displayed at the terminal.

### FORMAT:

$$[n[,m]] \left\{ \begin{array}{l} \text{SEARCH} \\ \text{S} \end{array} \right\} \text{"search\_expression"}$$

### ARGUMENTS:

[n]

Line number at which to begin the search. If you do not specify n, search begins at the current cursor position.

[,m]

Line number at which to end the search. If you do not specify m and have specified n, search ends at the last line of the file.

"

(Delimiter) Can be any character. You must use the same character in the two locations where a delimiter is required. If using a delimiter that is a character within the search expression, you must use the special character !C before the character within the text. It is recommended that you use a delimiter that is not within the search expression.

## SEARCH

search\_expression

String of characters that is the object of the search.

Example 1:

S "ABC"

Search for the first occurrence after the current cursor position of the string ABC.

Example 2:

S /AB"C/

Search for the first occurrence after the current cursor position of the character string AB"C.

Example 3:

S BAB

Search for the first occurrence after the current cursor position of A. Since the first non-blank character after the directive (B) is used as the delimiter, the search expression is that character string found between the first and second occurrence of the character B.

Example 4:

S ""

Search for the next occurrence of the previously defined search expression.

## SEARCH\_BACKWARD

### SEARCH BACKWARD (SEARCH BACKWARD OR SB)

Search the buffer from the current cursor position back to line 1 for the specified search expression.

The cursor is positioned on the first character of the matched search expression. The line containing the match is displayed on the first line of the window.

If you have previously defined a search expression, simply entering the directive followed by the two identical delimiters searches for the next occurrence of the search expression.

Searching begins at the current cursor position and continues backwards, from right to left, toward line 1 of the buffer until a match is found. If no match is found, the message SEARCH FAILED is displayed.

#### FORMAT:

```
{ SEARCH_BACKWARD } "search_expression"  
{ SB }
```

#### ARGUMENTS:

"

(Delimiter) Can be any character. You must use the same character in the two locations where a delimiter is required. If using a delimiter that is a character within the search expression, you must use the special character !C before the character within the text. It is recommended that you use a delimiter that is not within the search expression.

search\_expression

String of characters that is the object of the search.

## SEARCH\_BACKWARD

Example 1:

SB "ABC"

Search for the first occurrence before the current cursor position of the string ABC.

Example 2:

SB "AB!C"C"

Search for the first occurrence before the current cursor position of the string AB"C.

Example 3:

SB BAB

Search for the first occurrence before the current cursor position of A. Since the first non-blank character after the directive (B) is used as the delimiter, the search expression is that character string found between the first and second occurrence of the character B.

## SEARCH\_FORWARD

### SEARCH FORWARD (SEARCH FORWARD OR SF)

Search the buffer from the current cursor position to the end of the buffer for the specified search expression.

The cursor is positioned on the first character of the matched search expression. The line containing the match is displayed on the first line of the window.

If you have previously defined a search expression, simply entering the directive followed by the two identical delimiters searches for the next occurrence of the search expression.

Searching begins at the current cursor position and continues forward, from left to right, toward the last line of the buffer until a match is found. If no match is found, the message SEARCH FAILED is displayed.

#### FORMAT:

```
{ SEARCH_FORWARD } "search_expression"  
{ SF }
```

#### ARGUMENTS:

"

(Delimiter) Can be any character. You must use the same character in the two locations where a delimiter is required. If using a delimiter that is a character within the search expression, you must use the special character !C before the character within the text. It is recommended that you use a delimiter that is not within the search expression.

search\_expression

String of characters that is the object of the search.

## SEARCH\_FORWARD

Exampe 1:

```
SF "ABC"
```

Search for the first occurrence after the current cursor position of the character string ABC.

Example 2:

```
SF XAB"CX
```

Search for the first occurrence after the current cursor position of the character string AB"C.

Example 3:

```
SF BAB
```

Search for the first occurrence after the current cursor position of A. Since the first non-blank character after the directive (B) is used as the delimiter, the search expression is that character string found between the first and second occurrence of the character B.

Example 4:

```
SF ""
```

Search for the next occurrence of the previously defined search expression.



## TOP\_LINE

### TOP LINE (TOP LINE OR TL)

Display the first line (line 1) of the buffer at the top of the current window.

The cursor is positioned on line 1 in the column in which it was positioned before you executed the directive.

FORMAT:

{ TOP\_LINE }  
{ TL }

Example:

TL

Display the first line of the buffer at the top of the window.

# TRAILING\_BLANKS

## TRAILING BLANKS (TRAILING BLANKS OR TB)

Do not suppress trailing blanks on the lines within the buffer when text is written to a file.

A line with trailing blanks is a line in which some number of characters (at least one) at the end of a line are spaces. If you do not specify this directive, these spaces are lost (discarded) when the line is written to a file. If you do specify this directive, the Screen Editor preserves them.

Once entered, this directive remains in effect until the end of the Screen Editor session.

FORMAT:

{ TRAILING\_BLANKS }  
{ TB }

Example:

TB

Do not suppress trailing blanks when you write the buffer to a file.

## UPPER\_CASE

### UPPER CASE (UPPER CASE OR UC)

Convert all lowercase characters in a previously defined block to uppercase characters.

If the block contains characters within apostrophes (') or quotation marks ("), these characters are not converted.

You must have previously defined a block before you can use this directive. See the Block function key description later in this section for information on defining blocks.

FORMAT:

```
{ UPPER_CASE }  
{ UC }
```

Example:

Assume you have defined the following block:

This program calculates the weekly gross and net pay

Enter the Upper Case directive:

```
UC
```

The block now reads:

```
THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY
```

# VERSION

## VERSION (VERSION OR V)

Display the current Screen Editor version number.

This directive is informational only; the displayed version number appears in the message region of the screen.

FORMAT:

{ VERSION }  
{ V }

Example:

V

Display the current Screen Editor version number. For example, the message may read

SCORPEO 09/04/1121

where 09 is the month, 04 is the day, and 1121 is the time associated with the date the SCORPEO bound unit was created.

## WINDOW\_WIDTH

### WINDOW WIDTH (WINDOW WIDTH OR WW)

Set the window width for the window.

The width of a window is a measure of how far the right margin of the text can be. The Right Margin directive positions the right margin to the window's width. The Window Right function key moves the window until the window width is column 80 of the display.

The Window Width directive defines the widest part of the records of a file that can be seen by moving the window to the right. It is independent of the record length of the input file. No data is lost when the window width is smaller than the length of the lines read from a file. However, the parts of the lines that are beyond the window width cannot be seen until the window width is made wider.

When you call the Screen Editor (see "Loading the Screen Editor" earlier in this section), the window width is initialized to 256 characters except for COBOL (.C) files which are initially restricted to 80 characters (through their window width can be changed by WW).

FORMAT:

$\left. \begin{array}{l} \text{WINDOW\_WIDTH} \\ \text{WW} \end{array} \right\} \quad [\text{length}]$

ARGUMENT:

[length]

Maximum length in characters (bytes) of the window.  
Specify a decimal number from 80 to 256.

Default: 80 characters (for the WW directive, not the initial setting).

Example:

WW 132

Set the window width to 132 characters.

# WRITE

## WRITE (WRITE OR W)

Save the specified lines of the buffer in a file.

If you specify a file pathname of a file which already contains text, that text is overwritten.

### NOTE

If you write text out to a file other than the file which you are editing, the file of reference changes to the pathname specified in this directive.

### FORMAT:

[n[,m]]    { WRITE }    [path]  
              { W }

### ARGUMENTS:

None or any number of the following control arguments may be entered:

[n[,m]]

The starting line number (n) and the ending line number (m) of the text to be placed into a file. If you do not specify starting and ending line numbers, all lines in the buffer are written to the file. If you specify only a starting line number, all lines in the buffer beginning with that line to the end are written to the file.

If you do not specify a pathname, do not specify line numbers.

[path]

Pathname of the file that is to contain the specified lines of text. Can be any valid form of pathname. If you do not specify a file pathname, the text is placed in the current file whose name you specified when you called the Screen Editor or when you read in a file (see the Read directive earlier in this section). This file is called the file of reference. If you specify a file pathname of a file that does not currently exist, the file is created for you. If the file does not exist, the Screen Editor creates a variable sequential file with a control interval size of 512 bytes and a maximum record size of 256 bytes. If you do not specify a path, the default pathname used is the current file reference.

Example 1:

```
W ^VOL1>DIR>INVNTRY
```

Write the contents of the buffer to a file whose pathname is ^VOL1>DIR>INVNTRY.

Example 2:

Assume you called the Screen Editor as follows:

```
SCORPEO ^VOL1>DIR>INVNTRY
```

After editing the file, you wish to write all lines back to the same file. Enter the directive

```
W
```

Example 3:

```
10,20 W ^VOL1>DIR>INV_NEW
```

Write the contents of lines 10 through 20 to the file named ^VOL1>DIR>INV\_NEW.

# WRITE\_BLOCK

## WRITE BLOCK (WRITE BLOCK OR WB)

Write the specified block of text into a file.

You must have previously defined a block of text using the Block function key (described later in this section under "Function Keys").

You cannot write a block of text to the file you are currently editing (i.e., the file of reference).

If you specify a file pathname of a file which already contains text, that text is overwritten.

### NOTE

If you write text out to a file other than the file which you are editing, the file of reference changes to the pathname specified in this directive.

Use this directive to "save" a piece (block) of the buffer in a file.

See "Block Description" earlier in this section for details on blocks.

### FORMAT:

$\left. \begin{array}{l} \text{WRITE\_BLOCK} \\ \text{WB} \end{array} \right\} \text{ path}$

### ARGUMENT:

path

Pathname of the file that is to contain the block of text. Can be any valid form of pathname. You must specify a pathname different from the pathname of the file whose name you specified when you called the Screen Editor or when you read in a file (see the Read directive earlier in this section). If you specify a file pathname of a file that does not currently exist, the file is created for you. If the file currently exists, the new text overwrites the file's current contents. Do not specify the current file of reference.



Example:

Assume you have already defined a block of text such as:

```
CONST
  FEDTAX   = 0.05;
  STATAXLO = 0.04;
  STATAXHI = 0.07;
```

You want to write this block of text to a file named  
^VOL1>DIR>PAY whose contents are:

```
PROGRAM PAY (INPUT,OUTPUT);
(*THIS PROGRAM CALCULATES THE WEEKLY GROSS AND
  NET PAY OF AN UNDETERMINED NUMBER OF EMPLOYEES*)
```

By entering the directive line

```
WB ^VOL1>DIR>PAY
```

the contents of the file ^VOL1>DIR>PAY are now:

```
CONST
  FEDTAX   = 0.05;
  STATAXLO = 0.04;
  STATAXHI = 0.07;
```

FUNCTION KEYS

On a general purpose keyboard, the function keys are on the top row and are numbered F1, F2, etc. On data entry and word processing keyboards these keys are described with other text. When a function key is pressed, the Screen Editor performs the action associated with that key. Each Screen Editor function key may have two definitions: one for normal (unshifted) depression, and one for depression with the SHIFT key.

The keyboard design differs, depending on the kind of keyboard you have. Figures 4-3 through 4-8 summarize the Screen Editor function keys and associates them with their proper function key by keyboard. The function keys are:

- Append Line
- Backward Word
- Block
- Copy Block
- Delete Block
- Erase Block
- Forward Word
- Move Block
- Window Down
- Window Left
- Window Right
- Window Up.

These function keys are described in detail on the following pages.

|                       |                 |                  |                 |               |                 |                |
|-----------------------|-----------------|------------------|-----------------|---------------|-----------------|----------------|
|                       | F1              | F2               | F3              | F4            | F5              | F6             |
| S<br>H<br>I<br>F<br>T |                 |                  |                 | COPY<br>BLOCK | ERASE<br>BLOCK  | APPEND<br>LINE |
|                       |                 | BACKWARD<br>WORD | FORWARD<br>WORD | MOVE<br>BLOCK | DEFINE<br>BLOCK |                |
|                       | F7              | F8               | F9              | F10           | F11             | F12            |
| S<br>H<br>I<br>F<br>T | DELETE<br>BLOCK | WINDOW<br>LEFT   | WINDOW<br>RIGHT |               |                 |                |
|                       |                 | WINDOW<br>UP     | WINDOW<br>DOWN  | QUIT          |                 | HELP           |

86-142

Figure 4-3. Screen Editor Template for VIP780X General Purpose Keyboard

|       | F1 | F2            | F3           | F4         | F5           | F6               |
|-------|----|---------------|--------------|------------|--------------|------------------|
| SHIFT |    |               |              | COPY BLOCK | ERASE BLOCK  | APPEND LINE      |
|       |    | BACKWARD WORD | FORWARD WORD | MOVE BLOCK | DEFINE BLOCK | INSERT CHARACTER |

|       | F7               | F8          | F9           | F10  | F11 | F12  |
|-------|------------------|-------------|--------------|------|-----|------|
| SHIFT | DELETE BLOCK     | WINDOW LEFT | WINDOW RIGHT |      |     |      |
|       | DELETE CHARACTER | WINDOW UP   | WINDOW DOWN  | QUIT |     | HELP |

86-140

Figure 4-4. Screen Editor Template for VIP730X and HDS 2 General Purpose and Data Entry Keyboard

|       | F1    | F2 | F3            | F4 | F5           | F6          |
|-------|-------|----|---------------|----|--------------|-------------|
| SHIFT | RESET |    |               |    | ERASE BLOCK  | APPEND LINE |
|       | QUIT  |    | DELETE TO EOL |    | DEFINE BLOCK | HELP        |

|       | F7           | F8          | F9           | F10 | F11           | F12          |
|-------|--------------|-------------|--------------|-----|---------------|--------------|
| SHIFT | DELETE BLOCK | WINDOW LEFT | WINDOW RIGHT |     |               |              |
|       |              | WINDOW UP   | WINDOW DOWN  |     | BACKWARD WORD | FORWARD WORD |

86-141

NOTE

BLACK INSERT KEY TOGGLES INSERT CHAR MODE.  
THE MOVE AND COPY KEYS ARE MARKED

Figure 4-5. Screen Editor Template for VIP7300 and VIP7800 Word Processing Keyboard

|         | F1    | F2 | F3 | F4    | F5     | F6             | F7             | F8   | F9     | F10           | F11          | F12           |
|---------|-------|----|----|-------|--------|----------------|----------------|------|--------|---------------|--------------|---------------|
| CONTROL |       |    |    |       |        |                |                |      |        |               |              |               |
| SHIFT   |       |    |    | COPY  | ERASE  | APPEND<br>LINE | BLOCK          | LEFT | RIGHT  |               |              |               |
| UNSHIFT | RESET |    |    | MOVE  | DEFINE |                | CHAR-<br>ACTER | UP   | DOWN   | BACK-<br>WARD | FOR-<br>WARD | TRANS-<br>MIT |
|         |       |    |    | BLOCK |        |                | DELETE         |      | WINDOW |               | WORD         |               |

86-143

Figure 4-6. Screen Editor Template for microSystem 6/10 Keyboard

|       | F1             | F2              | F3               | F4            | F5              | F6              | F7              |
|-------|----------------|-----------------|------------------|---------------|-----------------|-----------------|-----------------|
| SHIFT | WINDOW<br>LEFT | WINDOW<br>RIGHT | BACKWARD<br>WORD | COPY<br>BLOCK | ERASE<br>BLOCK  | APPEND<br>LINE  | DELETE<br>BLOCK |
|       | WINDOW<br>UP   | WINDOW<br>DOWN  | FORWARD<br>WORD  | MOVE<br>BLOCK | DEFINE<br>BLOCK | INSERT<br>CHAR. | DELETE<br>CHAR. |

86-144

Figure 4-7. Screen Editor Template for VIP7200 Keyboard

|       | F1             | F2              | F3               | F4            | F5              | F6             | F7              |
|-------|----------------|-----------------|------------------|---------------|-----------------|----------------|-----------------|
| SHIFT | WINDOW<br>LEFT | WINDOW<br>RIGHT | BACKWARD<br>WORD | COPY<br>BLOCK | ERASE<br>BLOCK  | APPEND<br>LINE | DELETE<br>BLOCK |
|       | WINDOW<br>UP   | WINDOW<br>DOWN  | FORWARD<br>WORD  | MOVE<br>BLOCK | DEFINE<br>BLOCK |                |                 |

86-145

Figure 4-8. Screen Editor Template for VIP7201 Keyboard

# APPEND LINE

## APPEND LINE

Append a new line after the line on which the cursor is positioned.

The new line appears as a blank line on which you can enter text. You must position the cursor in the text region of the screen for the Append Line function key to take effect.

To insert a line before the first line of text (line 1 in the file), position the cursor on the control line, and press the Append Line function key. Enter the new text on the blank line that is displayed.

You cannot insert lines before the Screen Editor control line.

The Append Line function key performs the same actions as the INS LINE key described later in this section.

Example:

```
(*****  
  THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY
```

Press the Append Line function key and the text appears as follows:

```
(*****  
  THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY
```

You can now enter text on the blank line on which the cursor is resting. Be aware that all line numbers following the new line are incremented by one.

# BACKWARD WORD

## BACKWARD WORD

Position the cursor from its current position to the first character of the previous word. A word is considered a string of characters delimited by blanks.

If the cursor is positioned in the middle of a word, it is repositioned to the beginning of that word.

If the cursor is positioned on the first word of a line, it is repositioned to the first character of the last word of the previous line.

Example 1:

The current cursor position is:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY

Press the Backward Word function key. The cursor is now positioned as follows:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY

Example 2:

The current cursor position is:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY  
OF AN UNDETERMINED NUMBER OF EMPLOYEES.

Press the Backward Word function key. The cursor is now positioned as follows:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY  
OF AN UNDETERMINED NUMBER OF EMPLOYEES.

## BLOCK

### BLOCK

Designate the first and last positions of a block of text.

Position the cursor on the character that is the first character of a block of text on which you wish to perform some action. Press the Block function key; this defines the beginning of the block. Position the cursor on the last character of the text you are defining as a block of text. Press the Block function key again; this defines the end of the block. The beginning and end of a block can be defined in any order.

A block is defined by its location; i.e., SCORPEO retains the line number and the column position of the first character of the block, and the line number and column position of the final character of the block. The content of the block is all the data within these two end points. As data within the block is altered, the block definition is not changed. This can cause an implicit change in the content of the block. For example, if a block contains lines 20 through 60 and line 50 is deleted, the block still contains lines 20 through 60. However, once line 50 is deleted, the text following the deleted line moves up to fill the space. The line that was line 61 before the deletion now becomes line 60, the final line of the block.

The definition of a block remains in effect until you use it, or cancel the block by pressing the Erase Block function key (described later in this section).

You can define only one block at a time.

You must define a block before using any of the block function keys (Erase Block, Delete Block, Copy Block, and Move Block; all are described later in this section), or any of the block directives (Write Block and Change Block) defined earlier in this section.

Example:

Locate the block of text you wish to define:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY

Position the cursor on the character that is the first character of the block you wish to define, in this case T. Press the Block function key.

## BLOCK

Next, position the cursor on the character that is the last character of the block you wish to define, in this case Y. Press the Block function key.

The block you have just defined is:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY



# COPY BLOCK

## COPY BLOCK

Copy a defined block of text to a specified position.

You must have previously defined a block before attempting to copy it. See the Block function key description and "Block Description" earlier in this section for information on defining blocks.

When a block is copied, the text of the defined block is:

- Retained in its original position
- Replicated at the current cursor position
- Inserted to the left of the current cursor position.

The results of a Copy Block operation depend on whether or not the cursor is on the first character of the line when the Copy Block function key is pressed.

- If the cursor is on the first character of the line, each line of the block is copied as a new line. The line where the cursor was originally positioned remains unchanged and follows the last line of the copied block.
- If the cursor is not on the first character of the line, the first line of the block is appended to the right of the character that was to the immediate left of the original cursor position. Any intermediate lines of the block are appended next. The final line of the block is inserted to the left of the character that was under the original cursor position; this character and all characters to its right are preserved.

Do not attempt a Copy Block operation when the cursor is positioned beyond the last line of the file being edited.

After the Copy Block function key is used, the cursor is positioned on the first character of the newly inserted text.

Example:

The previously defined block is:

WEEKLY NET PAY

COPY BLOCK

You wish to copy the block into the position specified by the arrow:

```
*****  
THIS PROCEDURE CALCULATES  
THE WEEKLY GROSS PAY  
AND ←  
OF AN UNDETERMINED NUMBER OF EMPLOYEES
```

Position the cursor on the second space character on the line beginning with AND. Press the Copy Block key.

The text row reads:

```
*****  
THIS PROCEDURE CALCULATES  
THE WEEKLY GROSS PAY  
AND WEEKLY NET PAY  
OF AN UNDETERMINED NUMBER OF EMPLOYEES
```

The block of text remains in its original position and is copied into the new position.

# DELETE BLOCK

## DELETE BLOCK

Delete the previously defined block of text.

You must have previously defined a block before attempting to delete it. See the Block function key description for information on defining blocks. After the Delete Block function key is used, the cursor is positioned on the first character that remains after the last character of the deleted block. The line containing the cursor is the first line in the window.

You do not need to position the cursor on the originally defined block to delete it.

The definition of the block is erased after using Delete Block function key.

See "Block Description" earlier in this section for details on blocks.

Example:

Assume you have previously defined the block designated by the shaded rectangles:

```
THIS IS AN EXAMPLE TO SHOW WHAT HAPPENS  
WHEN TEXT IS DELETED BY USING  
THE DELETE BLOCK FUNCTION KEY WITH  
SPLIT LINES.
```

Press the Delete Block function key. The text now reads:

```
THIS IS AN EXAMPLE WITH  
SPLIT LINES.
```

# ERASE BLOCK

## ERASE BLOCK

Cancel the definition of the previously defined block.

You must have previously defined a block (or have partially defined a block) before attempting to erase it. See the Block function key description for information on defining blocks.

There is no effect on the text within the defined block; only the block definition is cancelled.

You do not need to position the cursor on the originally defined block to erase it.

Example:

Assume you have previously defined the following block:

AND WEEKLY NET PAY

Press the Erase Block function key to cancel the definition of this block.

## FORWARD WORD

### FORWARD WORD

Position the cursor on the first character of the next word after the current cursor position.

A word is considered a string of characters delimited by spaces.

If the cursor is on the last word of a line, the cursor is positioned on the first character of the first word of the following line.

Example:

The current position of the cursor is:

```
SWT := STATAXLO * GROSSPAY;
```

Press the Forward Word function key. The new cursor position is:

```
SWT := STATAXLO * GROSSPAY;
```

# MOVE BLOCK

## MOVE BLOCK

Move a previously defined block of text to a specified position.

You must have previously defined a block before attempting to move it. See the Block function key description and "Block Description" earlier in this section for information on defining blocks.

When a block is moved, the text of the defined block is:

- Deleted from its original position
- Replicated at the current cursor position
- Inserted to the left of the current cursor position.

The results of a Move Block operation depend on whether or not the cursor is on the first character of the line when the Move Block function key is pressed.

- If the cursor is on the first character of the line, each line of the block is copied as a new line. The line where the cursor was originally positioned remains unchanged and follows the last line of the copied block.
- If the cursor is not on the first character of the line, the first line of the block is appended to the right of the character that was to the immediate left of the original cursor position. Any intermediate lines of the block are appended next. The final line of the block is inserted to the left of the character that was under the original cursor position; this character and all characters to its right are preserved.

Do not attempt a Move Block operation when the cursor is positioned beyond the last line of the file being edited.

After the Move Block function key is used, the cursor is positioned on the first character of the newly inserted text.

Example:

The previously defined block is:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY

You wish to move the block into the position specified by the arrow.

```
*****  
←  
OF AN UNDETERMINED NUMBER OF EMPLOYEES.
```

Position the cursor on the space where you want the first character of the block to appear:

```
*****  
█  
OF AN UNDETERMINED NUMBER OF EMPLOYEES.
```

Press the Move Block function key. The text now reads:

```
*****  
THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY  
OF AN UNDETERMINED NUMBER OF EMPLOYEES.
```

The block of text is deleted from its original position.

# WINDOW DOWN

## WINDOW DOWN

Move the current window toward the last line of the buffer by the number of lines specified in the scroll amount (see the Scroll Change directive earlier in this section for information on scroll amounts).

Scrolling stops at the last line in the buffer.

If you append lines after the last line in the buffer, the window is automatically moved down to accommodate the appended lines.

Default: 18 lines.

Example:

Assume the scroll amount is set at 5 and the text in the window appears as:

```
(10) IF TOTWRK1 >= STRTIME
(11) THEN
(12) SUBGROSS := RATE * TOTWRK1;
(13) SUBGROSS := SUBGROSS * PENNYRND;
(14) SUBGROSS := ROUND (SUBGROSS);
(15) SUBGROSS := SUBGROSS " PENNYRND;
      .
      .
      .
(28) END
```

Press the Window Down function key. The current window now displays:

```
(15) SUBGROSS := SUBGROSS " PENNYRND;
      .
      .
      .
(32) FWT := FWT " PENNYRND;
(33) WRITE (FWT :9:2);
```



## WINDOW LEFT

### WINDOW LEFT

Move the current window 40 columns to the left (toward the left margin) of the buffer.

Use this function key when you have entered text beyond column 80 and you wish to view text entered before column 1 of the current window.

If the current window is already at the left margin (displays column 1), no action is taken.

Example:

Assume the following phrase begins in column 81:

ENTERED BEFORE COLUMN 1 OF THE CURRENT WINDOW

Press the Window Left function key. The text that begins 40 characters before column 81 now appears in the window:

OND COLUMN 80 AND YOU WISH TO VIEW TEXT ENTERED BEFORE COL...

# WINDOW RIGHT

## WINDOW RIGHT

Move the current window 40 columns to the right (toward the right margin) of the buffer.

Use this function key when you have entered text beyond column 80 and you wish to view text entered beyond the last column of the current window.

If you enter text beyond column 80 of the current window, the window automatically moves 40 columns to the right.

If the current window already displays the text at the right margin, no action is taken.

### Example:

Assume the window begins in column 1 and the last 54 characters displayed are:

USE THIS FUNCTION KEY WHEN YOU HAVE ENTERED TEXT BEYOND

Press the Window Right function key. The text that begins 40 characters beyond the first column of the current window is displayed. The last 54 characters displayed are now:

RED TEXT BEYOND COLUMN 80 AND YOU WISH TO VIEW TEXT ENTERED B

# WINDOW UP

## WINDOW UP

Move the current window toward the first line of the buffer by the number of lines specified as the scroll amount (see the Scroll Change directive described earlier in this section for information on scroll amounts).

Scrolling stops at the first line of the buffer.

Example:

Assume the scroll amount is set at 5 and the text in the current window appears as:

```
(15) SUBGROSS := SUBGROSS " PENNYRND;  
      .  
      .  
      .  
(32) FWT := FWT " PENNYRND;  
(33) WRITE (FWT :9:2);
```

Press the Window Up function key. The current window now displays:

```
(10) IF TOTWRK1 <= STRTIME  
(11) THEN  
(12) SUBGROSS := RATE * TOTWRK1;  
(13) SUBGROSS := SUBGROSS * PENNYRND;  
(14) SUBGROSS := ROUND (SUBGROSS);  
(15) SUBGROSS := SUBGROSS " PENNYRND;  
      .  
      .  
      .  
(28) END
```

## LABELED KEYS

Labeled keys perform the functions described on the key. Depending on the type of terminal you are using you may or may not have these labeled keys. If your terminal does not have the listed labeled key, one of the function keys performs the same action. Function keys are described earlier in this section. Labeled key descriptions are listed alphabetically by key name on the following pages. Those labeled keys that have corresponding function keys are identified in the labeled key description. The labeled keys listed below are described in the following subsections, and are based on the VIP7801/VIP7802 (general purpose) terminal:

- BACKSPACE
- CARRIAGE RETURN/CR/ENTER NEW LINE/RETURN
- CLEAR/RESET
- CTL/CTRL CLR/TAB/SET
- CTL/CTRL TAB
- CURSOR DOWN (↓)
- CURSOR LEFT (←)
- CURSOR RIGHT (→)
- CURSOR UP (↑)
- DEL CHAR
- DEL LINE
- ERASE EOL
- HOME
- INS CHAR
- LINE FEED
- TAB CTL I/CTRL I
- TAB CLR
- TAB SET.

### NOTE

The labeled keys AUTO LF (Automatic Line Feed) and LOCAL can cause unpredictable results during a screen editing session.

AUTO LF causes an automatic line feed each time you press carriage return. The Screen Editor then performs another line feed/carriage return. This results in double spacing and loss of the correct cursor position.

LOCAL allows you to move the cursor on the screen without interrupting processing. Using this key causes loss of the correct cursor position unless the cursor is repositioned to its original location before you exit from local mode.

Use of either of these keys is not recommended during screen editing sessions.

Table 4-2 correlates the VIP7801/VIP7802 (general purpose) terminal with the VIP7301 (general purpose ) keyboard, the VIP7303/VIP7803 (word processing) keyboards, the VIP7307 (data entry) keyboard, the microSystem 6/10 Workstation (VIP7305), the VIP7200 keyboard, and the VIP7201 keyboard. The slash character (/) used in the table represents the word or.

Table 4-2. Correlation of SCORPEO's Labeled Keys

| VIP7801/VIP7802  | VIP7301          | VIP7303/VIP7803 | VIP7307       | microSystem<br>6/10 Workstation<br>(VIP7305) | VIP7200          | VIP7201      |
|------------------|------------------|-----------------|---------------|--|------------------|--------------|
| BACKSPACE        | BACKSPACE        | BACKSPACE       | CTRL B        | Backspace                                    | CTL H            | BACKSPACE    |
| RETURN           | RETURN           | RETURN          | FIELD ENTER   | Return/CR                                    | RETURN           | RETURN       |
| CLEAR/RESET      | CLEAR/RESET      | SHIFT CLEAR     | NUM CLEAR     | Reset  | SHIFT CLEAR      | SHIFT CLEAR  |
| CTRL CLR/TAB/SET | CTRL CLR/TAB/SET | CTRL T          | CTRL T        | Control Init                                 | CTL Y            | CTRL T       |
| CTRL TAB         | CTRL TAB         | CTRL TAB        | CTRL TAB      | Shift  | CTL B            |              |
| DEL CHAR         | FCN KEY 7        | DELETE          | DELETE        | Delete                                       | SHIFT FCN KEY F7 | DELETE       |
| DEL LINE         | SHIFT DEL        | SHIFT DELETE    | NUM DEL       | Shift Delete                                 | DEL              | SHIFT DELETE |
| ERASE EOL        | ERASE            | ERASE           | CTRL E        | Erase  | ERASE            | EOL          |
| HOME             | HOME             | HOME            | FCN KEY 1     | Home   | HOME             | HOME         |
| INS CHAR         | FCN KEY 6        | INSERT          | INSRT         | Insert                                       | SHIFT FCN KEY F6 | INSERT       |
| INS LINE         | SHIFT FCN KEY 6  | SHIFT INSERT    | NUM FCN KEY 6 | Shift Insert                                 | FCN KEY F6       | SHIFT INSERT |
| LINE FEED        | LINE FEED        | UNSUPPORTED     | UNSUPPORTED   | LF   | LF               | LINE FEED    |
| TAB              | TAB              | TAB             | TAB           |  | CTL I            |              |
| TAB CLR          | TAB CLR          | SHIFT ABBREV.   | MDFY          | Control Clear                                | CTL C            | CTRL C       |
| TAB SET          | TAB SET          | ABBREV          | FORM          | Control Set                                  | CTL S            | CTRL S       |

Some of the keys on the microSystem 6/10 Workstation Multifunction (MF) keyboard are unlabeled. For definition of these keys, use the SCORPEO Keyboard Overlay (T2010).

# BACKSPACE

## BACKSPACE

Move the cursor one position (character) to the left.

If the cursor is positioned on the leftmost column showing on the screen, pressing BACKSPACE has no effect.

If the current window does not display the left margin, and the cursor is positioned in column 1 of the current window, pressing the BACKSPACE key automatically moves the window 40 columns to the left.

Example:

Assume the cursor is resting on a line of text as follows:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY

Press the BACKSPACE key. The cursor is now positioned as follows:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY

**CARRIAGE RETURN  
CR  
ENTER  
NEW LINE  
RETURN**

CARRIAGE RETURN

Move the cursor from its current position to the leftmost column (column 1) of the succeeding line.

Scrolling occurs, bringing the leftmost column into the window, if necessary.

If the cursor is positioned on the last line of the window, the window automatically moves to display the next nine lines in the buffer.

Example:

Assume the cursor is positioned as follows:

```
IF TOTWRK1 >= STRTIME
THEN
SUBGROSS := RATE * TOTWRK1;
SUBGROSS := SUBGROSS * PENNYRND;
```

Press CARRIAGE RETURN. The cursor is now positioned as follows:

```
IF TOTWRK1 <= STRTIME
THEN
SUBGROSS := RATE * TOTWRK1;
SUBGROSS := SUBGROSS * PENNYRND;
```



## CLEAR/RESET

### CLEAR/RESET

Cancel the character string just entered on the line.

#### NOTE

If transmission errors occur or if some other type of data corruption occurs on the screen, pressing the CLEAR/RESET key redisplay the entire screen (the three regions).

#### Example:

Assume you have just entered the following line and the cursor is positioned as shown:

```
THIS PROGRAM CALUCLATE
```

Rather than using function keys and labeled keys to correct the errors, press the CLEAR/RESET key. The line of text is cancelled, and the cursor is positioned at the beginning of the same line.

**CTL CLR/TAB/SET**  
**CTRL CLR/TAB/SET**

CTL CLR/TAB/SET  
CTRL CLR/TAB/SET

Cancel all user-defined tab stops.

Example:

Assume you have defined tab stops (using the Language Type directive) at columns 8, 12, 21, and then every 10 columns.

Pressing the key sequence CTL CLR/TAB/SET cancels all tab stops, leaving the TAB key undefined.

**CTL TAB  
CTRL TAB**

CTL TAB  
CTRL TAB

Move the cursor back one tab stop from its current position according to the currently defined tab stops.

Example:

With the default tab stops set, the current cursor position is:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY

Press the CTL TAB sequence. The cursor is now positioned as follows:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY

# CURSOR DOWN

## CURSOR DOWN ( ↓ )

Move the cursor down one line leaving the cursor in the same column.

If the cursor is positioned on the last line of the window, pressing the Cursor Down key positions the cursor on the first line of the window. The column is unchanged. To append blank lines to the end of the file, move the cursor past the current end of the file. A blank character (giving a line length of one) is appended each time the cursor is moved to a new line.

Example:

Assume the cursor is positioned as follows:

```
IF TOTWRK1 <= STRTIME
THEN
SUBGROSS := * TOTWRK1;
SUBGROSS := SUBGROSS * PENNYRND;
```

Press the Cursor Down key. The cursor is now positioned as follows:

```
IF TOTWRK1 <= STRTIME
THEN
SUBGROSS := RATE * TOTWRK1;
SUBGROSS := SUBGROSS * PENNYRND;
```

## CURSOR LEFT

### CURSOR LEFT (←)

Move the cursor one position (character) to the left.

If the cursor is positioned on the leftmost column on the screen, pressing the Cursor Left key moves the cursor to the rightmost character on the screen of the preceding line.

If the cursor is positioned in the leftmost column of line 1 of the current window, pressing the Cursor Left key moves the cursor to the rightmost column of the last line displayed in the window.

Example:

Assume the cursor is positioned as follows:

```
IF TOTWRK1 <= STRTIME
THEN
SUBGROSS := RATE * TOTWRK1;
SUBGROSS := SUBGROSS * PENNYRND;
```

Press the Cursor Left key. The cursor is now positioned as follows:

```
IF TOTWRK1 <= STRTIME
THEN
SUBGROSS := RATE * TOTWRK1;
SUBGROSS := SUBGROSS * PENNYRND;
```

## CURSOR RIGHT

### CURSOR RIGHT (→)

Move the cursor one position (character) to the right of its current position.

If the cursor is positioned on the rightmost column on the screen, pressing the Cursor Right key moves the cursor to the leftmost character on the screen of the succeeding line. Moving the cursor past the end of the line on which the cursor is situated causes blanks (spaces) to be appended to the end of the line. To keep these trailing blanks, use the TB option when the text is written to a file.

If the cursor is in the rightmost column of the last line in the window, pressing the Cursor Right key positions the cursor in column 1 of the first line in the window. If the cursor is positioned at the rightmost edge of the window, inserting one or more characters at this point moves the cursor to column 81. This situation causes all lines in the window to move to the left by 40 columns, and positions the cursor in column 41 (now the leftmost column on the screen).

Example:

Assume the cursor is positioned as follows:

```
IF TOTWRK1 <= STRTIME
THEN
SUBGROSS := RATE * TOTWRK1;
SUBGROSS := SUBGROSS * PENNYRND;
```

Press the Cursor Right key. The cursor is now positioned as follows:

```
IF TOTWRK1 <= STRTIME
THEN
SUBGROSS := RATE * TOTWRK1;
SUBGROSS := SUBGROSS * PENNYRND;
```

## CURSOR UP

### CURSOR UP ( ↑ )

Move the cursor up one line leaving the cursor in the same column.

If the cursor is positioned on the first line of the window, pressing the Cursor Up key positions the cursor on the last line of the window. The column is unchanged.

If you have positioned the cursor in the Directive Region of the screen (by pressing HOME), pressing the Cursor Up key returns the cursor to its location before you pressed the HOME key. (The HOME key is described later in this section.)

Example:

Assume the cursor is positioned as follows:

```
IF TOTWRK1 <= STRTIME
THEN
SUBGROSS := RATE * TOTWRK1;
SUBGROSS := SUBGROSS * PENNYRND;
```

Press the Cursor Up key. The cursor is now positioned as follows:

```
IF TOTWRK1 <= STRTIME
THEN
SUBGROSS := RATE * TOTWRK1;
SUBGROSS := SUBGROSS * PENNYRND;
```

# DEL CHAR

## DEL CHAR

Delete the character on which the cursor is positioned.

To delete a character, position the cursor on the unwanted character and press the DEL CHAR key. The line that contained the deleted character is now one character shorter in length. All characters following the deleted character are moved one position to the left so that the first character following the deleted character is adjacent to the character preceding the deleted character.

To delete multiple characters, press and hold the DEL CHAR key.

Example:

Assume the text reads as follows:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY

Position the cursor on the unwanted character:

THIS PRO~~O~~GRAM CALCULATES THE WEEKLY GROSS AND NET PAY

Press the DEL CHAR labeled key. The text now reads:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY



## DEL LINE

### DEL LINE

Delete the line on which the cursor is positioned.

After the line is deleted, the cursor is positioned in the same column but on the line that immediately followed the deleted line.

Example:

Assume the cursor is resting on the line as follows:

```
IF TOTWRK1 <= STRTIME
THEN
SIB
SUBGROSS := RATE * TOTWRK1;
SUBGROSS := SUBGROSS * PENNYRND;
```

Press the DEL LINE labeled key. The text now reads as follows:

```
IF TOTWRK1 <= STRTIME
THEN
SUBGROSS := RATE * TOTWRK1;
SUBGROSS := SUBGROSS * PENNYRND;
```

# ERASE EOL

## ERASE EOL

Delete any text from (and including) the current cursor position to the end of the line.

To erase characters from the current cursor position to the end of the line, position the cursor on the first character of the text you want to erase, and press the ERASE EOL key. If the cursor is on column 1 of the line, the ERASE EOL does not totally delete the line, but modifies the line length to one character, a blank.

The cursor is positioned on the same line in the same column as when you pressed the ERASE EOL key.

Example:

Assume the text reads as follows and you have positioned the cursor as shown:

THIS PROGRAM CALCULATES THE GROSS AND NET PAY OF AN OF  
AN UNDETERMINED NUMBER OF EMPLOYEES.

Press the ERASE EOL key. The text now reads:

THIS PROGRAM CALCULATES THE GROSS AND NET PAY OF  
AN UNDETERMINED NUMBER OF EMPLOYEES.

HOME

Move the cursor to the directive input line of the screen.

When you press the HOME key, the system "remembers" the cursor position before it is positioned to the directive input line. The "remembered" cursor position is the cursor position used for any directives related to cursor position.

Example:

Assume the cursor is positioned as follows:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY

Press the HOME key. The cursor position is remembered and the cursor is repositioned to the directive input line as follows:

DIRECTIVE:█

To cancel a directive, press the Cursor Up key--do not press RETURN. The Cursor Up key returns the cursor to its "remembered" position.

# INS CHAR

## INS CHAR

Insert a number of characters to the left of some point in a buffer.

To insert characters, position the cursor to the character that should immediately follow the character(s) that you are inserting. Press the INS CHAR key. The flag INSERT appears in the status region to alert you that you are in insert mode. Enter the new characters.

Every character to the right of the current cursor position (including the character on which the cursor is resting) is moved one space to the right for the insertion of each character you enter.

To end character insertion, press the INS CHAR key a second time. The INSERT flag is removed from the status region. Any characters you enter now will write over the existing text.

To insert characters at the end of a line, position the cursor to the location where you wish to begin the insert and simply enter the characters. It is not necessary to use the INS CHAR key to enter characters at the end of a line.

When SCORPEO is in insert mode, the RETURN and LINE FEED keys can be used to split one line into two lines or create a blank line. While in insert mode, if you press RETURN and enter it as an inserted character, the character where the cursor is located, and all the characters that follow the cursor on that same line, are moved to the next line and begin in column 1. If you now press RETURN with the cursor positioned in column 1, a blank line is inserted in the buffer, and the original line moves to the next line.

While in insert mode, if you press LINE FEED and enter it as an inserted character, the character where the cursor is located, and all the characters that follow the cursor on that same line, are moved to the next line, with each character remaining in its original column. The cursor is positioned on the first character of the new line (it remains in the same column, one line down). If you are not in insert mode and press LINE FEED, the new line is blank filled up to the cursor position.

## Example 1:

An example of pressing RETURN in insert mode is:

THIS IS A SPLIT LINE.

Press RETURN. The result is:

THIS IS A  
SPLIT LINE.

## Example 2:

An example of pressing LINE FEED in insert mode is:

THIS IS A SPLIT LINE.

Press LINE FEED. The result is:

THIS IS  
A SPLIT LINE.

To insert multiple characters of the same value, press and hold the INS CHAR key. If inserting characters in a line causes the maximum line length of 256 characters to be exceeded, those characters in the rightmost columns of the line (past column 256) are lost.

## Example 3:

Position the cursor to the right of the location where you want to insert characters. For example:

NETPAY := GROSSPAY - FICA;

Press the INS CHAR key. Enter the characters you wish to insert. The new line of text (after insertion) now reads:

NETPAY := GROSSPAY - FWT - SWT - FICA;

Press the INS CHAR key again to leave insert mode.

# INS LINE

## INS LINE

Append a new line after the line on which the cursor is positioned.

The new line appears as a blank line on which you can enter text. You must position the cursor in the text region of the screen for the INS LINE key to take effect.

To insert a line before the first line of text (line 1 in the file), do the following:

1. Position the cursor on the control line (you cannot insert lines before the Screen Editor control line)
2. Press the INS LINE key.

Enter the new text on the blank line that is displayed. Pressing RETURN after the new line of text inserts another blank line below the previous new line. Enter the next line of new text on this line. To exit INS LINE mode, use the Cursor Up (↑) or Cursor Down (↓) key--do not press RETURN at the end of the last line.

The INS LINE key performs the same actions as the Append Line function key described earlier in this section.

Example:

Position the cursor on the line before the location of the line of text you want to insert:

```
(*****  
  THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY  
*****)
```

Press the INS LINE key and the text appears as follows:

```
(*****  
  THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY  
*****)
```

You can now enter text on the blank line on which the cursor is positioned. Be aware that all line numbers following the new line are incremented by one.

## LINE FEED

### LINE FEED

Move the cursor down one line from its current position and leave the cursor in the same column.

If the cursor is positioned on the last line of the window, pressing the LINE FEED key moves the window down nine lines.

Example:

Assume the cursor is positioned as follows:

```
IF TOTWRK1 <= STRTIME
THEN
SUBGROSS := RATE * TOTWRK1;
SUBGROSS := SUBGROSS * PENNYRND;
```

Press the LINE FEED key. The cursor is now positioned as follows:

```
IF TOTWRK1 <= STRTIME
THEN
SUBGROSS := RATE * TOTWRK1;
SUBGROSS := SUBGROSS * PENNYRND;
```

**TAB  
CTL I  
CTRL I**

TAB  
CTL I  
CTRL I

Move the cursor from its current position to the next tab stop to the right on that line, according to the current tab stop definition.

The key sequences CTL I or CTRL I perform the same actions as the TAB key.

Example:

With the default tab stops set, the current cursor position is:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY

Press the TAB key. The cursor is now positioned as follows:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY



## TAB CLR

### TAB CLR

Cancel (clear) the tab stop definition in a specified column.

Position the cursor in the column containing the tab stop and press the TAB CLR key.

Example:

Assume you have set the tab stops as shown by the designated cursor positions:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY

Positioning the cursor in the column containing the tab stop you wish to cancel (say, the first cursor position) cancels the definition of that tab stop:

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NET PAY

# TAB SET

## TAB SET

Set a tab stop in a specified column.

Position the cursor in the desired column and press the TAB SET key.

Example:

Assume this line begins in column 1. At each designated cursor position, you have pressed the TAB SET key.

THIS PROGRAM CALCULATES THE WEEKLY GROSS AND NEY PAY

Tab stops are now set at columns 9 and 19.

REMOVE THIS PAGE AND PLACE TAB FOR

TAB 5

LINE EDITOR



## *Section 5*

# ***LINE EDITOR***

This section describes Line Editor functions and the Line Editor directive set. Procedural information on using Line Editor directives to create and edit files is also included at the end of this section.

### OVERVIEW

The Line Editor creates and/or alters character text that constitutes files; the files usually are source unit files. The statements in a source unit file can be written in any of the programming languages supported by MOD 400. Throughout this section, it is assumed that source unit files are being edited.

Editing is controlled by directives entered to the Line Editor through the device specified in the in path argument of the Enter Group Request (EGR) command. This device can be reassigned in the command that loads the Line Editor.

All editing is done in a temporary work area called the current buffer. When the Line Editor is invoked, the Line Editor creates a current buffer. To save Line Editor output, you must write the source unit contents of the current buffer to a file.

During a single execution of the Line Editor, the Line Editor can operate in input and/or edit mode. In input mode, you can create a source unit and/or add one or more specified lines to an existing source unit. In edit mode, you can locate and change single characters, words, or a string of characters, read the contents of a file into the current buffer so that the line(s) can be edited, write lines from the current buffer to a file, and terminate execution of the Line Editor.

#### NOTES

1. During a single execution of the Line Editor, you can create and/or change any number of files. You must delete the contents of the current buffer before you begin to edit another file, unless you want that file to comprise the same information that was in the previous file(s).
2. At any time during execution of the Line Editor you can request a message that indicates whether input or edit mode is in effect. Each time !? is entered, the following message is issued:

INPUT MODE  
EDIT

Line Editor processing can be interrupted by either:

- Pressing the QUIT, INTERRUPT, or BREAK key on your terminal
- Entering  $\Delta C \Delta B$ group-id on the operator terminal, where group-id is the two-character group identification code associated with the group containing the task to be interrupted.

A **\*\*BREAK\*\*** message appears on your terminal when the system interrupts the Line Editor. If the Program Interrupt (PI) command is entered, output is suppressed and the task returns to directive input level. See the Commands manual for a detailed description of the Break function.

Each Line Editor directive's name and function is listed in Table 5-1. They are described in detail in "Input Mode Description and Directives," "Edit Mode Description and Directives," and "Advanced Usage of the Line Editor" later in this section. Directives described in the input and edit mode subsections operate within the current buffer.

## LINE EDITOR SUFFIX CONVENTIONS

During the program preparation, it is convenient to identify output file(s) with the name of the input file.

When you create a source unit, you should append the appropriate suffix identification character to the name of the file that will contain the source unit. The suffix designates the type of text that constitutes the source unit. The suffix must be .C for COBOL or C programs, .F for FORTRAN programs, .B for BASIC programs, .PS for Pascal programs, .AS for ADA, and .A for Assembly language programs.

When you specify the file names of Line Editor input and output files (in Line Editor directives), the editor requires that you designate the complete file name, including the suffix that denotes the contents of the file. The Line Editor does not append a suffix to its input and output files.

## LINE EDITOR DIRECTIVE FORMAT CONVENTIONS

Most Line Editor directives consist of only a directive name, a directive name preceded by one or two addresses, or a directive name optionally preceded by one or two addresses and followed by text and termination escape characters (!F) that designate the end of the directive and cause the Line Editor to switch from input mode to edit mode. These formats are illustrated here. Note that if a directive includes text, the text may be specified beginning immediately after the directive name (see Format 4) or beginning on the next line (see Format 5).

FORMAT 1:

dirname

FORMAT 2:

adr<sub>1</sub> dirname

FORMAT 3:

adr<sub>1</sub> { ; } adr<sub>2</sub> dirname

FORMAT 4:

$\left[ \text{adr}_1 \left[ \left\{ \begin{array}{l} ; \\ , \end{array} \right\} \text{adr}_2 \right] \right] \text{dirname}[\text{text}]!F$

FORMAT 5:

$$\left[ \begin{array}{c} \text{adr}_1 \quad \left\{ \begin{array}{c} ; \\ , \end{array} \right\} \text{adr}_2 \\ \text{[text]} \end{array} \right] \text{dirname}$$

.  
.  
.  
!F

ARGUMENTS:

dirname

Valid Line Editor directive.

adr<sub>1</sub> adr<sub>2</sub>

Valid addresses for the current buffer.

text

Any text.

NOTES

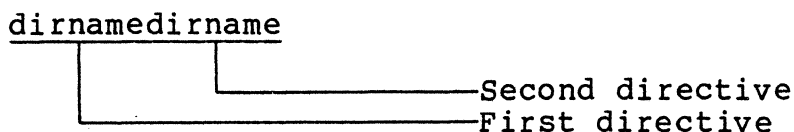
1. Spaces are not permitted, except in the following circumstances:
  - a. Spaces are permitted in expressions constituting addresses.
  - b. A space is permitted after the Execute, Read, and Write directive names (these directives are described later in this section).
2. One or two addresses may be specified without a directive name; if no directive name is specified, the last (or only) addressed line is printed (see "Print (P)" later in this section).

When a single address is specified, the Line Editor locates the specified line in the current buffer. When two addresses are specified within a single directive, the Line Editor locates a specified series of lines in the current buffer; the lines that are located depend on whether the addresses are separated by a comma or a semicolon (see "Referencing a Series of Lines"). If a Line Editor directive format designates that either a single address or a pair of addresses may be entered, you can enter that directive and omit one or both addresses; their default value(s) will be used. Address default values are described later in this section under each directive's argument descriptions.

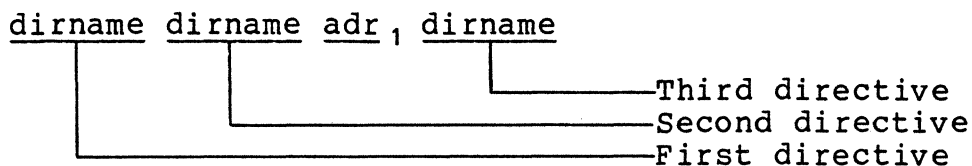


Multiple Line Editor directives can be entered on a single line; it is not necessary to separate each directive with a delimiter, but one or more spaces can be specified, as illustrated below:

Directives not separated by delimiters:



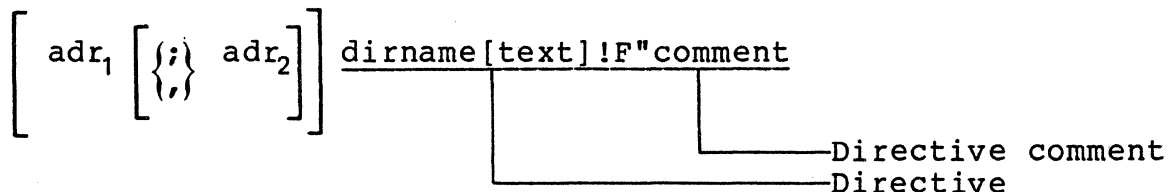
Directives separated by delimiters:



A comment can be included at the end of a directive line (i.e., at the end of the last or only directive); the comment must be preceded by a quotation mark ("), as illustrated:

adr dirname dirname"comment

To include a comment after an input mode directive, specify the comment after the terminator !F; otherwise, the comment is included as text.



If a terminal is the directive input device, press RETURN at the end of each line.

### Methods of Specifying Addresses

Each address can be specified by one of the following methods or by a combination of these methods:

- Number of line
- Position of line relative to the "current" line
- Contents of line.

## DESIGNATING A LINE NUMBER AS AN ADDRESS

Each line in the current buffer can be located by a decimal number that indicates the current position of the line within the buffer. The first line in the buffer is line 1; subsequent lines are numbered sequentially in ascending order. To determine the line number of a specified line in the current buffer, enter the Print Line Number directive. To determine the line number and contents of specified line(s) in the buffer, enter the Print With Line Number directive. (These directives are described under "Advanced Usage of the Line Editor" later in this section.) Multiple decimal numbers separated by plus or minus signs can be specified to represent a line number.

Example:

```
10
5+5
```

Each of the expressions above request line 10. The last line can be referenced by its line number or by the character \$.

If an address designates a line that is not in the current buffer, an error message is issued.

Line Editor directives can cause lines to be added to or deleted from the current buffer. Each time this occurs, all succeeding lines are renumbered. For example, if line 15 is deleted, line 16 becomes 15, and each subsequent line number is decremented by 1.

## DESIGNATING THE POSITION OF A LINE RELATIVE TO THE "CURRENT" LINE AS AN ADDRESS

Most Line Editor directives affect either the current line or a line a designated number of positions from the current line. If the last Line Editor directive entered was an Input directive (i.e., input mode was in effect), the current line is the last line added or read by the Line Editor (regardless of whether the condition specified in the directive was met). If the last Line Editor directive entered was an Edit directive (edit mode was in effect), the current line is the last line of text edited. The current line can be located by specifying a period (.).

### NOTE

If you do not know which line is the current line, you can obtain a display of the line number of the current line by specifying the Print Line Number directive, which is described under "Advanced Usage of the Line Editor" later in this section.

You can locate lines relative to the current line by specifying an address that consists of a period followed by one or more signed decimal numbers. For example, the address `.+1` specifies the line immediately following the current line, the address `.-1` specifies the line immediately preceding the current line, and `.+5+5-3` specifies the seventh line after the current line.

When specifying an increment to the current line number, you can omit the plus (+) sign; e.g., `.5` is interpreted as `.+5`. When specifying a decrement to the current line number, you can omit the period; e.g., `-3` is interpreted as `.-3`.

#### DESIGNATING CONTENTS OF LINE AS AN ADDRESS

You can designate that the Line Editor locate the first line that contains a specified character or a specified sequence of characters by designating those characters in an expression as an address. An expression comprises one or more ASCII characters, which must be delimited by slashes (e.g., `/ASCII characters/`).

The Line Editor searches the lines in the current buffer until it finds the first occurrence of the specified expression; unless specified otherwise, the expression can be in any position within the line. If a circumflex (^) is designated as the first character of the expression, the expression must be the first expression on the line. If \$ is designated as the last character of the expression, the expression must be the last expression on the line. (Use of these special characters is described in the following paragraphs.) The Line Editor searches from the line immediately following the current line (i.e., `.+1`) through the last line in the buffer; if a line containing the specified expression is not found, the Line Editor then searches line 1 to the current line. In the directive format:

`/BBB/dirname`

the address is the expression BBB. The Line Editor searches as many lines as necessary for the first occurrence of BBB. The contents of the source unit being searched are listed below. (The numbers in parentheses represent line numbers.)

- (1) AAA
- (2) BBB
- (3) CCC (current line)
- (4) BBB

The specified directive causes the Line Editor to locate line number 4, since this is the first line after the current line that contains the expression BBB.

When the following ASCII characters are included in expressions, they have special meanings:

| <u>Character</u>                      | <u>Description</u>  |
|---------------------------------------|---|
| *                                     | Requests expressions that contain any number (or none) of the immediately preceding character(s).   |
| ^                                     | When designated as the <u>first</u> character of an expression, requests lines that begin with the specified expression (excluding the character ^).  |
| \$                                    | When specified as the <u>last</u> character of an expression, requests lines that end with the specified expression (excluding the character \$).   |
| .                                     | Can be any character on any line; specify one period per character (e.g., ".." means any two characters on any line).   |
| &                                     | Can be used in the string expression of a Substitute directive to indicate that the strings of characters preceding and following "&" are to be concatenated to the target string of the search. See the description of the Substitute directive later in this section. |
| line feed<br>(hex 0A)<br>(see Note 3) | The occurrence of a line feed in the string expression determines the point in the resulting line at which the line is to be split into two lines. See the Substitute directive for further details.  |

#### NOTES

1. The special meanings of the above characters, / (which delimits an expression) and !? (which causes display of the mode currently in effect), can be removed by preceding the special character with !C. For example, !C!?! causes !? to be interpreted as text rather than as a request for display of the mode that is in effect.
2. The characters . and \$ can be specified as line numbers or as special characters in expressions; the Line Editor can interpret their meaning from the way they are used.

3. For the Line Editor, two hexadecimal characters can be interpreted as one ASCII byte by using the escape sequence !Hxx, where xx are the two hex characters. However, this feature must be used with care since some of the hexadecimal characters may be confused with control or special characters in ASCII strings. The following is a list of the hexadecimal characters whose use is restricted:

0A is the line feed character; in a string expression, it is interpreted as a request for advancement to a new line.

2E in a regular expression is treated as ".".

26 in a string expression is treated as "&".

2A in a regular expression is treated as "\*".

24 at the end of a regular expression is interpreted as "end-of-line (\$)".

5E at the beginning of a regular expression becomes "beginning-of-line (^)".

Rather than attempting to substitute in an expression using the characters above, execute a Change directive, reentering the line using hexadecimal and ASCII characters for the entire line.

Following are some examples of expressions specified as addresses in Line Editor directives. Following each expression is a description of the line/character(s) in the current buffer for which the Line Editor searches. In each case, the Line Editor searches the lines sequentially, starting with the line immediately following the current line to the end of the file, and then from line 1 through the current line.

| <u>Expression</u> | <u>Description</u>   |
|-------------------|--|
| /A/.              | Locates the first line that contains the expression A in any position in that line.                |
| /ABC/             | Locates the first line that contains the expression ABC in any position on that line.              |
| /AB*C/            | Locates the first line that contains the expression AC or A followed by any number of B's and a C. |

| <u>Expression</u> | <u>Description</u>   |
|-------------------|--|
| /IN..TO/          | Locates a line that contains IN and TO separated by any two characters.  |
| /IN.*TO/          | Locates a line that contains IN and TO, in that order, with any or no characters between those two words.                              |
| /^ABC/            | Locates a line that begins with the expression ABC.  |
| /ABC\$/           | Locates a line that ends with the expression ABC.  |
| /ABC!C\$/         | Locates a line that contains the expression ABC\$. ABC\$ can be in any character positions, since the character \$ was preceded by !C. |
| /^ABC.*DEF\$/     | Locates a line that begins with ABC and ends with DEF; there can be any number of characters between ABC and DEF.                      |

The Line Editor remembers the last specified expression. That expression can be reinvoked in a subsequent Line Editor directive by specifying a null expression (e.g., //).

Examples:

/ABC/dirname

Expression ABC specified as address

2dirname

Second line in buffer specified as address

//dirname

Specifies ABC as an address, since ABC was the last specified expression

An address can be specified as an expression followed by one or more signed decimal integers. Each of the following three expressions requests the second line after the line that contains ABC.

/ABC/2

/ABC/+2

/ABC/+5-3

## COMPOUND ADDRESSES

An address can be formed by combining any of these methods. If a compound address contains a line number, the line number must be the first element of the address.

The first element of the compound address determines the starting location from which the Line Editor searches for the designated expression. If the first element is a line number, the Line Editor searches for the expression starting with the line that immediately follows the specified line number. (Ordinarily, the Line Editor searches starting with the line that immediately follows the current line.)

Example 1:

```
10/ABC/
```

The Line Editor searches the lines in the current buffer for the characters ABC, starting with line 11.

Example 2:

```
.-8/ABC/
```

The Line Editor searches the lines in the current buffer for the character ABC, starting eight lines before the current line.

Example 3:

```
/ABC/DEF/
```

The Line Editor searches for the first line containing DEF that occurs after the first line containing ABC.

Each expression in a compound address can be followed by a signed decimal integer.

Example 4:

```
/ABC/-10/DEF/5
```

The Line Editor searches for the first occurrence of the character string DEF that is within 10 lines before the first line that contains ABC. After DEF is found, the current line is the fifth line after the line containing the match for DEF.

## Referencing a Series of Lines

A Line Editor directive that permits two addresses to be specified causes the Line Editor to locate a series of lines in the buffer. The addresses can be separated by a comma or a semicolon. If the second address is relative to the current line (plus or minus), both the addresses and the plus or minus sign determine which lines will be located by the Line Editor; otherwise, only the addresses are relevant.

If the addresses are separated by a comma, the Line Editor locates the line at the first address through the line at the second address, inclusive. The current line remains unchanged until the directive is executed; the current line then becomes the line specified by the second address.

If the addresses are separated by a semicolon, the line located by the first address becomes the current line and the value of the second address is calculated.

Example 1:

```
1,5dirname
```

These addresses specify lines 1 through 5, inclusive. After the directive is executed, line 5 becomes the current line.

Example 2:

```
1,$dirname
```

These addresses specify line 1 through the last line in the buffer, inclusive. After the directive is executed, the last line becomes the current line.

Example 3:

```
.1,/ABC/
```

These addresses specify the line immediately following the current line through the first line that contains ABC. The first line that contains ABC then becomes the current line.

Example 4:

```
.1,.2dirname
```

The contents of a sample source unit are listed below. The numbers in parentheses represent line numbers.



- (1) ABC
- (2) DEF (current line)
- (3) GHI
- (4) ABC
- (5) XYZ
- (6) ABC

These addresses specify the line immediately following the current line through the second line after the current line. The Line Editor locates lines 3 and 4. Line 4 becomes the current line.

Example 5:

```
.1;2dirname
```

These addresses are the same as those in Example 4, but they are separated by a semicolon. If the contents of the sample source unit are the same as in Example 4, this directive causes the Line Editor to locate lines 3, 4, and 5. This first address specifies the line immediately after the current line; i.e., line 3. Line 3 then becomes the current line. The second address specifies that the Line Editor locate through the second line after the (new) current line; i.e., lines 4 and 5.

The same series of lines can be requested by specifying their addresses in more than one way, using different delimiters.

Example 6:

```
/ABC/,/ABC/+3dirname  
/ABC/;.+3dirname
```

The contents of a sample source unit follows. The numbers in parentheses represent line numbers.

- (1) ABC
- (2) DDD (current line)
- (3) EEE
- (4) FFF
- (5) GGG
- (6) HHH

The first series of addresses specifies that the Line Editor locate the first line that contains ABC (line 1) through the third line after that line (lines 2, 3, and 4). Line 4 becomes the current line.

The second series of addresses specifies that the Line Editor locate the first line that contains ABC (line 1), make that line the current line, and then reference three lines from the new current line (lines 2, 3, and 4). Line 4 becomes the current line.

## Loading the Line Editor

The Line Editor command loads the Line Editor. Upon loading, a message indicating the current Line Editor release number is sent to the error-out file.

To load the Line Editor, enter the ED command.

FORMAT:

```
ED[?SILENT] [ctl_arg]
```

ARGUMENTS:

```
[?SILENT]
```

Optional entry point that suppresses the welcome message.

```
[ctl_arg]
```

None or any number of the following control arguments can be entered:

-IN path

File from which Line Editor directives are to be read. -IN path in the Line Editor command line results in the user-in file being changed to path or the contents of path being copied to a buffer (EXEC). Execution starts with the first line of (EXEC). If the path is relative, it is expanded relative to the current working directory.

Default: directives are obtained from the current user-in file.

```
{-LINE_LEN nnn }  
{-LL nnn }
```

Alter the line length to be acted upon by the Line Editor. Can be any value from 20 to 512. Default: nn equals 512.

```
{-PROMPT }  
{-PT }
```

Print the prompt character E? (in edit mode) or I? (in input mode) on the user-in file upon completion of the previous Line Editor directive; no carriage return follows. If the user-in file is other than a terminal-like device, this argument is ignored.

{ -NO BLANK\_SUPPRESS }  
{ -NBS }

No blank suppression; i.e., the Line Editor does not suppress trailing blanks on the input line (for one invocation only). Subsequent invocation without -NBS suppresses trailing blanks.

{ -FILE\_SIZE nn }  
{ -FS nn }

Alter the initial size of the work file to the size in the user-supplied value of nn, where nn is a decimal integer comprising up to four characters and designates the number of 256-byte control intervals. If an output file is created, it is initialized to the same size.

Default: 4.

{ -ARGS strings }  
{ -ARG strings }

Up to nine character strings that are numbered sequentially and can be passed to the Line Editor in the Editor directive. (See "Change Origin of Text During Edit Mode (!B)" later in this section.) Each argument following the -ARG keyword is copied to buffer (ARGn). "n" denotes the position of the argument following the -ARG and can be any value from one through nine. If specified, this argument and its strings must be entered last.

{ -SAFE name }  
{ -SF name }

Permanent work files called name.EDWK1 and name.EDWK2 are created in your current working directory to contain the two latest copies of the current buffer. Name can be from one to six characters. Abnormal termination causes the work files to be closed in their current state and saved for later use, and normal termination releases them. To reuse the work files, invoke the Line Editor without -SAFE or with -SAFE and a different name, read each one in, and continue editing the one that contains the latest copy of the saved buffer.

Default: Work files are temporary files and are released under all conditions. If the Line Editor terminates abnormally, all modifications made after the last Write directive are lost.

{ -SIZE nn }  
 { -SZ nn }

Define the number of 1024-word blocks to be used for dynamic storage in memory. "nn" can be any value from 1 to 64. The formula for calculating the number of lines possible is  $(2*nnK/LL+6)-3$ , where K is 1024 and LL is the line length value (or 80 by default).

Default: 1.

SUMMARY OF LINE EDITOR DIRECTIVES AND ESCAPE SEQUENCES

Table 5-1 lists each Line Editor directive name and escape sequence, summarizes its function, and designates the topic in this section under which the directive/escape sequence is described. The topics refer to the following paragraphs:

- "Input Mode Description and Directives" (input mode)
- "Edit Mode Description and Directives" (edit mode)
- "Advanced Usage of the Line Editor"
  - "General Advanced Line Editor Directives" (advanced usage--general)
  - "Auxiliary Buffer Directives and Escape Sequences" (advanced usage--auxiliary buffers)
  - "Line Editor Debugging Directives" (advanced usage--debugging)
  - "Line Editor Programming Directives" (advanced usage--programming).

Table 5-1. Summary of Line Editor Directives and Escape Sequences

| Directive Name/Escape Sequence | Function  | Topic Under Which Described                                 |
|--------------------------------|---|---|
| A                              | Add line(s) after specified address.                | Append directive (input mode)                               |
| B                              | Make specified auxiliary buffer the current buffer. | Change Buffer directive (advanced usage--auxiliary buffers) |
| C                              | Delete specified line(s) and insert other line(s).  | Change directive (input mode)                               |

Table 5-1 (cont). Summary of Line Editor Directives and Escape Sequences

| Directive Name/Escape Sequence | Function  | Topic Under Which Described                          |
|--------------------------------|---|--|
| D                              | Delete specified line(s) from current buffer.   | Delete directive (edit mode)                         |
| E                              | Execute command other than Line Editor without exiting from the Line Editor.  | Execute directive (advanced usage--general)          |
| G                              | Search for specified line(s) that contain specified character string.   | Global directive (advanced usage--general)           |
| I                              | Add line(s) <u>before</u> a specified address.  | Insert directive (input mode)                        |
| K                              | Copy line(s) in current buffer to specified auxiliary buffer. Do <u>not</u> delete lines from current buffer. Overlay existing line(s) in auxiliary buffer. | Copy directive (advanced usage--auxiliary buffers)   |
| L                              | Send line feed to the user-out file.  | Line Feed directive (advanced usage--general)        |
| M                              | Move line(s) from current buffer to specified auxiliary buffer; delete the lines from current buffer and overlay existing line(s) in auxiliary buffer.      | Move directive (advanced usage--auxiliary buffers)   |
| N                              | Designate different line as the current line.   | New Current Line directive (advanced usage--general) |
| P                              | Print specified line(s) in current buffer.  | Print directive (edit mode)                          |
| Q                              | Conditionally terminate execution of Line Editor.   | Quit directive (edit mode)                           |

Table 5-1 (cont). Summary of Line Editor Directives and Escape Sequences

| Directive Name/Escape Sequence | Function   | Topic Under Which Described  |
|--------------------------------|--|--|
| R                              | Read text from file to current buffer.   | Read directive (edit mode)   |
| S                              | Substitute character string with another character string.                                 | Substitute directive (edit mode)   |
| T                              | Display line of text on user-out file. Subsequent input/output will be on the next line.   | Type directive (advanced usage--programming)                                     |
| U                              | Convert specified uppercase expression to lowercase.                                       | Lowercase directive (advanced usage--general)                                    |
| V                              | Search for specified line(s) that do <u>not</u> contain specified character string.        | Exclude directive (advanced usage--general)                                      |
| W                              | Write specified line(s) from current buffer to specified file.                             | Write directive (edit mode)  |
| X                              | Request status of auxiliary buffers.   | Buffer status directive (advanced usage--auxiliary buffers)                      |
| ZDUMP                          | Print contents of specified line(s).   | Hexadecimal dump directive (advanced usage--debugging)                           |
| ZREGEXP                        | Display last specified expression.   | ZREGEXP directive (advanced usage--debugging)                                    |
| ZTRACE                         | Display each directive line before it is executed.   | ZTRACE directive (advanced usage--debugging)                                     |
| !B                             | Change origin of text to specified auxiliary buffer or execute specified auxiliary buffer. | change origin of text during input/edit mode (advanced usage--auxiliary buffers) |

Table 5-1 (cont). Summary of Line Editor Directives and Escape Sequences

| Directive Name/Escape Sequence | Function  | Topic Under Which Described  |
|--------------------------------|---|--|
| !C                             | Remove meaning of following special character.  |  |
| !F                             | Terminate an input mode directive.  | (Input mode)   |
| !Hxx                           | Interpret two following hexadecimal characters as one ASCII byte.   |  |
| !K                             | Copy line(s) in current buffer to specified auxiliary buffer; do not delete existing line(s) in auxiliary buffer.   | Copy-append directive (advanced usage--auxiliary buffers)                      |
| !L                             | Send line feed to the error-out file.   | Line feed directive (advanced usage--general)                                  |
| !M                             | Move line(s) from current buffer to specified auxiliary buffer; delete the line(s) from current buffer and append them to existing line(s) in auxiliary buffer. | Move-append directive (advanced usage--auxiliary buffers)                      |
| !P                             | Type line number and contents of specified line(s) in current buffer.   | Print With Line Number directive (advanced usage--general)                     |
| !Q                             | Unconditionally terminate execution of Line Editor.   | Quit directive (edit mode)   |
| !R                             | Accept single line from terminal.   | Accept Single Line from Terminal directive (advanced usage--auxiliary buffers) |
| !T                             | Display line of text on user-out file; subsequent input/output will be on the same line.  | Type directive (advanced usage--programming)                                   |

Table 5-1 (cont). Summary of Line Editor Directives and Escape Sequences

| Directive Name/Escape Sequence | Function  | Topic Under Which Described                           |
|--------------------------------|---|---|
| !U                             | Convert specified lower-case expression to uppercase.                                 | Uppercase directive (advanced usage--general)         |
| !?                             | Cause message indicating whether input or edit mode is in effect.                     |   |
| #                              | If current buffer contains data, execute specified directive(s).                      | If Data directive (advanced usage--programming)       |
| address #                      | If current line is specified line, execute specified directive(s).                    | If Line directive (advanced usage--programming)       |
| !S                             | Replace each occurrence of specified character string with another character string.  | Substitute directive (edit mode)                      |
| addresses #                    | If current line is within specified lines, execute specified directive(s).            | If Range directive (advanced usage--programming)      |
| ^B                             | Release a specified auxiliary buffer.   | Destroy directive (advanced usage--auxiliary buffers) |
| ^ #                            | If current buffer does <u>not</u> contain data, execute specified directive(s).       | If Empty directive (advanced usage--programming)      |
| address ^ #                    | If current line is <u>not</u> specified line, execute specified directive(s).         | If Not Line directive (advanced usage--programming)   |
| addresses ^ #                  | If current line is <u>not</u> within specified lines, execute specified directive(s). | If Not Range directive (advanced usage--programming)  |
| *                              | If specified expression is within specified lines, execute specified directive(s).    | Search directive (advanced usage--programming)        |



Table 5-1 (cont). Summary of Line Editor Directives and Escape Sequences

| Directive Name/Escape Sequence | Function   | Topic Under Which Described                            |
|--------------------------------|--|--|
| ^*                             | If specified expression is not within specified lines, execute specified directive(s).     | Search Not directive (advanced usage--programming)     |
| :                              | Define location to which Line Editor can be directed for subsequent directive(s).          | Label directive (advanced usage--programming)          |
| =                              | Type line number of specified line in current buffer.                                      | Print Line Number directive (advanced usage--general)  |
| >                              | Accept subsequent directive(s) from specified location in current buffer or interactively. | Go To directive (advanced usage--programming)          |
| ?                              | If specified line is in current buffer, execute specified directive(s).                    | Address Prefix directive (advanced usage--programming) |
| "                              | Annotate Line Editor files.  | Comment directive (advanced usage--programming)        |

### CREATING A SOURCE UNIT

To create a source unit, perform the following steps listed. Input mode directives are described under "Input Mode Description and Directives." Each of the directives referenced is described under "Edit Mode Description and Directives."

1. Load the Line Editor. (See "Loading the Line Editor" earlier in this section.)
2. If there already are lines in the current buffer, clear the buffer by specifying: 1,\$D.
3. Enter the appropriate Input directive and text to be included.
4. Make changes, if necessary, by entering the appropriate Input and/or Edit directive(s).

5. Write the contents of the current buffer to a file by using the Write directive.
6. Exit from the Line Editor by entering the Quit directive (optional).

#### CHANGING AN EXISTING SOURCE UNIT

To change an existing source unit, perform the following steps. Input mode directives are described under "Input Mode Description and Directives." Each of the directives referenced is described under "Edit Mode Description and Directives" later in this section.

1. Load the Line Editor, if it is not already loaded. (See "Loading the Line Editor" earlier in this section.)
2. If there already are lines in the current buffer, delete unwanted lines by specifying the Delete directive.
3. Use the Read directive to read into the current buffer the source unit to be edited.
4. Enter the appropriate Edit and/or Input directive(s).
5. Write the contents of the current buffer to the file from which the lines were read or to a different file by using the Write directive.
6. Exit from the Line Editor by entering the Quit directive (optional).

#### INPUT MODE DESCRIPTION AND DIRECTIVES

During input mode, you can create a source unit or add lines to an existing source unit by entering through the directive input device one or more input directives.

Input directives have the following capabilities:

- Add lines after a specified address (Append directive).
- Delete specified lines and insert other specified lines (Change directive).
- Add lines before a specified address (Insert directive).

You can create a source unit by using the Append or Insert directive. You can add lines to an existing source unit by using any or all of the above directives.

Each input directive must have one of the following formats:

FORMAT 1:

$$\left[ \text{adr}_1 \left[ \begin{array}{c} \{;\} \\ \{,\} \end{array} \right] \text{adr}_2 \right] \text{dirname}$$

text

·  
·

!F["comment]

FORMAT 2:

$$\left[ \text{adr}_1 \left[ \begin{array}{c} \{;\} \\ \{,\} \end{array} \right] \text{dirname} \text{text} !F["comment] \right]$$

If directives are being entered through a terminal, the directive name can either be immediately followed by a carriage return, and then text (i.e., the lines to be included in the source unit) or directive name can be immediately followed by text, with additional lines of text (if any) added on subsequent lines. The text can be any number of lines of ASCII characters. The maximum number of characters per line is determined by the value specified in the `-LINE_LEN n` argument of the ED command. The last line of text must be followed by the escape sequence `!F` to terminate input mode. Otherwise, the next Line Editor directive is interpreted as additional text. The escape sequence `!F` can be entered at the end of the last line of text or in the first character position of the next line. The next directive can begin in the next character position or on the next line.

#### NOTES

1. To enter a blank from the operator terminal, as the first character on a line, precede it with an `!C` sequence.
2. The characters `!F` can be included as text by preceding them with `!C`; in this case, `!F` does not designate the end of the text.

Input directives are described in detail on the following pages. In the examples, numbers in parentheses are references to line numbers and do not appear in memory or in text.

# APPEND

## APPEND (A)

Move one or more specified lines into the current buffer after a specified address. If multiple lines are specified, they are put into the buffer in the order in which they were entered. The Append directive can be used to create a source unit or to add lines to an existing source unit.

After the Append directive is executed, the current line is the last line appended. The appended line(s) are given line numbers and subsequent lines, if any, are renumbered.

### FORMAT 1:

```
[adr]A
text
.
.
!F
```

### FORMAT 2:

```
[adr]Atext!F
```

### ARGUMENT:

#### adr:

Address of the line immediately after which the specified lines are inserted.

Default: Current line. If the buffer is empty, the current line is line number 0.

### NOTE

If you are creating a new source unit, there is no need to specify an address.

Example 1, Creating a New Source Unit:

In this example, the buffer is empty.

```
A
WWW
XXX
YYY
ZZZ
!F
```

This Append directive puts lines WWW, XXX, YYY, and ZZZ into the current buffer. Since the buffer is empty, it is not necessary to specify an address. The lines are inserted, in the order in which they were entered, starting at line 1. The lines put into the buffer constitute a new source unit which can then be edited and/or written to a file.

Example 2, Adding Lines to an Existing Source Unit:

```
/TTT/A
UUU
!F
3A
WWW
XXX
!F
```

These Append directives put line UUU into the buffer immediately after the first line that contains TTT, and lines WWW and XXX into the buffer immediately after the third line.

The contents of the buffer are:

- (1) TTT
- (2) VVV

After the first Append directive is executed, the buffer contains:

- (1) TTT
- (2) UUU (current line)
- (3) VVV

## APPEND

After the second Append directive is executed, the buffer contains:

- (1) TTT
- (2) UUU
- (3) VVV
- (4) WWW
- (5) XXX (current line)

# CHANGE

## CHANGE (C)

Delete a single line or a series of lines in the current buffer and then insert the text specified between the directive name and the insert terminator !F.

After the Change directive is executed, the current line is the last line of inserted text. The inserted line(s) are given line numbers and subsequent lines, if any, are renumbered.

FORMAT 1:

$$\left[ \text{adr}_1 \left[ \begin{array}{l} \{;\} \\ \{,\} \end{array} \text{adr}_2 \right] \right] \text{C}$$

text

·  
·

!F

FORMAT 2:

$$\left[ \text{adr}_1 \left[ \begin{array}{l} \{;\} \\ \{,\} \end{array} \text{adr}_2 \right] \right] \text{Ctext!F}$$

ARGUMENTS:

$\text{adr}_1$

Address of the first or only line to be deleted and replaced. Default: Current line.

$\text{adr}_2$

Address of the last line to be deleted and replaced. Default: Only the line identified by  $\text{adr}_1$  is deleted and changed.

### NOTE

If both  $\text{adr}_1$  and  $\text{adr}_2$  are omitted, only the current line is deleted and replaced.

## CHANGE

In the following examples, the contents of the current buffer are:

- (1) AAA
- (2) BBB
- (3) CCC (current line)
- (4) DDD
- (5) EEE

Example 1:

```
2C
XXX
YYY
!F
```

This Change directive deletes the second line and replaces it with lines XXX and YYY. Subsequent lines are renumbered.

After the Change directive is executed, the buffer contains:

- (1) AAA
- (2) XXX
- (3) YYY (current line)
- (4) CCC
- (5) DDD
- (6) EEE

Example 2:

```
/BBB/, .1C
XXX
YYY
ZZZ!F
```

This Change directive deletes the first line that contains BBB (line 2) through the line immediately after the current line (line 4) and replaces them with lines XXX, YYY, and ZZZ, respectively.

After the Change directive is executed, the buffer contains:

- (1) AAA
- (2) XXX
- (3) YYY
- (4) ZZZ (current line)
- (5) EEE



Example 3:

```
  .,5C      or      .,$C  
XXX         or      XXX  
!F          !F
```

Each of the Change directives above deletes the current line through line 5 and replaces them with a single line containing XXX.

After the change directive is executed, the buffer contains:

- (1) AAA
- (2) BBB
- (3) XXX (current line)

# INSERT

## INSERT (I)

Insert one or more specified lines into the current buffer before a specified address. If multiple lines are specified, they are inserted in the order in which they were entered.

The Insert directive can be used to create a source unit or to add lines to an existing source unit.

After the Insert directive is executed, the current line is the last line inserted. The inserted line(s) are given line numbers, and subsequent lines, if any, are renumbered.

### FORMAT 1:

```
[adr]I  
text  
:  
:  
:  
!F
```

### FORMAT 2:

```
[adr]Itext!F
```

### ARGUMENT:

adr

Address of the line immediately before which the specified line(s) are inserted. Default: Current line.

### NOTE

If you are creating a new source unit, there is no need to specify an address.

## Example 1:

In this example, the current buffer is empty.

```
I
AAA
BBB
CCC
DDD
!F
```

This Insert directive creates in the current buffer a new source unit comprising lines AAA, BBB, CCC, and DDD, respectively. The lines can then be edited and/or written to a file.

In Examples 2, 3, and 4, the contents of the current buffer are:

```
(1) AAA
(2) BBB
(3) CCC
(4) DDD (current line)
```

## Example 2:

```
-2I
XXX
!F
```

This Insert directive designates that a line containing XXX be inserted two lines before the current line.

After the Insert directive is executed, the current buffer contains:

```
(1) AAA
(2) XXX (current line)
(3) BBB
(4) CCC
(5) DDD
```

## Example 3:

```
/AAA/I
H!C!FH
KKK
!F
```

## INSERT

This Insert directive designates that lines H!FH and KKK be inserted into the current buffer immediately before the first line that contains AAA. Note that when !F is part of the text, it is preceded by !C; when !F delimits the last line of text, it is not preceded by !C.

After the Insert directive is executed, the buffer contains:

- (1) H!FH
- (2) KKK (current line)
- (3) AAA
- (4) BBB
- (5) CCC
- (6) DDD

Example 4:

```
I
XXX
!F
```

This Insert directive designates that a line containing XXX be inserted immediately before the current line.

After the Insert directive is executed, the current buffer contains:

- (1) AAA
- (2) BBB
- (3) CCC
- (4) XXX (current line)
- (5) DDD

## EDIT MODE DESCRIPTION AND DIRECTIVES

During edit mode you can create a source unit or edit an existing source unit.

Edit mode directives have the following capabilities:

- Delete specified line(s) from the current buffer (Delete directive)
- Print on the user-out file specified line(s) in the current buffer (Print directive)
- Terminate execution of the Line Editor (Quit directive)
- Read text from specified file into the current buffer (Read directive)
- Substitute a designated string of characters in specified line(s) with another specified string of characters (Substitute directive)

### NOTES

1. To edit an existing source unit, the Read directive must be previously specified.
2. Until you are familiar with the Line Editor, enter Print directives frequently so you can determine the status of the lines being edited.
3. To save the results of an edited or newly created source unit, you must specify the Write directive before you terminate execution of the Line Editor.

Most edit mode directives have one of the following formats:

FORMAT 1:

dirname["comment"]

FORMAT 2:

adr<sub>1</sub> dirname["comment"]

FORMAT 3:

$$\left[ \text{adr}_1 \left[ \left\{ \begin{array}{l} ' \\ ; \end{array} \right\} \text{adr}_2 \right] \right] \text{dirname}["\text{comment}"]$$

Edit mode directives are described alphabetically on the following pages. In the examples, numbers in parentheses are references to line numbers and do not appear in memory or in text.

# DELETE

## DELETE (D)

Delete a single line or consecutive lines from the current buffer.

After the Delete directive is executed, each subsequent line in the buffer is renumbered, and the current line is the line that immediately follows the last line deleted or the last line in the buffer if the previous "last line" was deleted.

### FORMAT:

$$\left[ \text{adr}_1 \left[ \begin{array}{c} \{ ; \} \\ \{ / \} \end{array} \right] \text{adr}_2 \right] D$$

### ARGUMENTS:

$\text{adr}_1$

Address of the first or only line to be deleted.  
Default: Current line.

$\text{adr}_2$

Address of the last line to be deleted. Default: Only the line identified by  $\text{adr}_1$  is deleted.

### NOTE

If both  $\text{adr}_1$  and  $\text{adr}_2$  are omitted, only the current line is deleted.

In the following examples, the contents of the current buffer are:

- (1) AAA
- (2) BBB (current line)
- (3) CCC
- (4) DDD
- (5) EEE

## DELETE

Example 1:

1,3D

This Delete directive deletes lines 1 through 3. After this Delete directive is executed, the current buffer contains:

- (1) DDD (current line)
- (2) EEE

Example 2:

/CCC/D

In this Delete directive, `adr` is CCC and `adr` is not specified, so the only line that is deleted is the first line that contains CCC. After this Delete directive is executed, the current buffer contains:

- (1) AAA
- (2) BBB
- (3) DDD (current line)
- (4) EEE

Example 3:

.,3D

This Delete directive deletes the current line through line 3. After this Delete directive is executed, the current buffer contains:

- (1) AAA
- (2) DDD (current line)
- (3) EEE

Example 4:

D

This Delete directive does not include any addresses so only the current line, line 2, is deleted. After this directive is executed, the current buffer contains:

- (1) AAA
- (2) CCC (current line)
- (3) DDD
- (4) EEE



# PRINT

## PRINT (P)

Print a single line or consecutive lines in the current buffer. You can specify the address(es) of the line(s) to be printed, or you can request a printout of the first line that contains a specified expression. The printout is issued to the user-out file; i.e., the file designated in the -OUT out\_path argument of the Enter Batch Request (EBR) or Enter Group Request (EGR) command, unless the file was reassigned in the File Out (FO) command. If the printout occurs on the operator terminal, each line of text is preceded by the group identification characters.

After the Print directive is executed, the current line is the last (or only) line printed.

### FORMAT 1:

Format including directive name P:

$$\left[ \text{adr}_1 \left[ \begin{array}{l} \{ ; \} \\ \{ , \} \end{array} \right] \text{adr}_2 \right] P$$

### FORMAT 2:

Format excluding directive name P:

$$\text{adr}_1 \left[ \begin{array}{l} \{ ; \} \\ \{ , \} \end{array} \right] \text{adr}_2$$

### ARGUMENTS:

adr<sub>1</sub>

Address of the first or only line to be printed. The Line Editor begins its search at the second line in the current buffer. Default: Current line.

adr<sub>2</sub>

Address of the last line to be printed. Default: Only the line identified by adr<sub>1</sub> is printed.

### NOTE

If both adr<sub>1</sub> and adr<sub>2</sub> are omitted and P is specified, only the current line is printed.

PRINT

In the following examples, the contents of the current buffer are:

- (1) AAABBB
- (2) CCCDDD (current line)
- (3) EEEFFF
- (4) GGGHHH

Example 1:

1,\$P

This Print directive causes a printout of each line in the current buffer.

AAABBB  
CCCDDD  
EEEFFF  
GGGHHH

After this directive is executed, the current line is line 4.

Example 2:

P

This Print directive causes a printout of only the current line.

CCCDDD

After this directive is executed, the current line still is line 2.

Example 3:

4P

This Print directive causes a printout of line 4.

GGGHHH

After this directive is executed, the current line is line 4.

Example 4:

```
.,4P
```

This Print directive causes a printout of the current line (line 2) through line 4:

```
CCDDDD  
EEEEFF  
GGGHHH
```

After this directive is executed, the current line is line 4.

Example 5:

```
/AAA/
```

This Print directive causes a printout of the first line that contains AAA.

```
AAABBB
```

After this directive is executed, the current line is line 1.

Example 6:

```
3D/AAA/
```

This example illustrates a directive line that contains both a Delete directive and a Print directive. This directive deletes line 3 and causes a printout of the first line that contains AAA. After the directives are executed, the current buffer contains:

```
(1) AAABBB  
(2) CCDDDD  
(3) GGGHHH
```

Line 1 prints out, and is the current line.

# QUIT

## QUIT (Q OR !Q)

Exit from the Line Editor. Quit must be specified at the end of the editing session. This directive must be the last or only directive on a line. If the directive input device is a terminal, the Quit directive must be immediately followed by a carriage return.

Quit is executed conditionally or unconditionally, depending on which Quit format is specified. In a conditional Quit request (Format 1), if there are any buffers which have been modified but not written to a file before the Quit directive is entered, a warning message is issued and Quit is not executed. After the message, any Line Editor directive(s), including Write, can be entered. If Write is not specified and Quit is reentered, the Quit directive is executed and changes specified in previous Line Editor directives are not saved. In an unconditional Quit request (Format 2), modified buffers are not checked before Quit is executed.

### FORMAT 1:

Q

### FORMAT 2:

!Q

### Example:

|                            |   |
|----------------------------|---|
| A                          | Append directive, which puts specified lines into current buffer. |
| AAABBB<br>CCCDDD<br>EEEEFF | Lines that are put into current buffer.                           |
| !F                         | Designate the end of the insertion.                               |
| 2D                         | Delete the second line of text (e.g., CCCDDD).                    |
| W FIRST                    | Write all lines in buffer to file named FIRST.                    |
| Q                          | Return control from the Line Editor to the Command Processor.     |

# READ

## READ (R)

Read text from a specified file into the current buffer. The Read directive must be the only or last directive on a line. After the Read directive is executed, the current line is the last line read from the file.

### FORMAT:

[adr]R [path]

### ARGUMENTS:

adr

Address of a line in the current buffer; the contents of the specified file are appended after this line. Default: Last line in the buffer; if the buffer is empty, the file is appended starting at the first line in the buffer.

path

Pathname of the ASCII file to be read into the current buffer. (Methods of specifying pathnames are described in Section 2.) The pathname can be preceded by any number of blanks. Default: Pathname specified in the latest Read or Write directive associated with the current buffer. To determine which pathname was specified last, specify the Buffer Status directive, which is described under "Advanced Usage of the Line Editor" later in this section. If the path argument is not specified and a pathname was not previously specified, an error message is issued.

### NOTE

!CDR or any other device name beginning with an exclamation point (!) can cause errors. The exclamation point is a Line Editor escape character. A read of !CDRxx (R !CDRxx) tries to read file name DRxx because !C is a cancel flag. Use >SPD> in place of the exclamation point (e.g., R >SPD>CDRxx), or cancel a C (e.g., R !C!CDRxx).

## READ

### Example 1:

R START

This Read directive reads into the current buffer the contents of a file whose simple pathname is START. Since an address is not specified, the lines are read into the buffer after the last line currently in the buffer.

The contents of START are:

- (1) AAA
- (2) BBB
- (3) CCC

If the buffer is empty, after the Read directive is executed, the current buffer contains:

- (1) AAA
- (2) BBB
- (3) CCC (current line)

If the buffer already contains:

- (1) XXX
- (2) YYY
- (3) ZZZ

After the Read directive is executed, the current buffer contains:

- (1) XXX
- (2) YYY
- (3) ZZZ
- (4) AAA
- (5) BBB
- (6) CCC (current line)

### Example 2:

/CCC/R NEW

This Read directive designates that the contents of the file whose simple pathname is NEW be read into the current buffer after the first line in the current buffer that contains CCC.

The contents of the current buffer are:

- (1) AAA
- (2) BBB (current line)
- (3) CCC
- (4) CCC

The contents of NEW are:

- (1) XXX
- (2) ZZZ

After the Read directive is executed, the current buffer contains:

- (1) AAA
- (2) BBB
- (3) CCC
- (4) XXX
- (5) ZZZ (current line)
- (6) CCC

Example 3:

This example illustrates the Read directive used in conjunction with Append and Write directives. The current buffer is empty.

```

A          Puts subsequent lines into the current buffer.
AAA
BBB
CCC
!F        Designates the end of the insert.
W NOW     Writes the contents of the current buffer to the
          file whose simple pathname is NOW.
R          Reads into the current buffer, after the last line
          in the buffer, the contents of NOW; NOW is the
          pathname specified in the last Write directive.

```

After the Read directive is executed, the current buffer contains:

- (1) AAA
- (2) BBB
- (3) CCC
- (4) AAA
- (5) BBB
- (6) CCC (current line)

# SUBSTITUTE

## SUBSTITUTE (S OR !S)

Replace each occurrence of a specified string of characters in a single line or in a sequence of lines with another specified string of characters.

After this directive is executed, the current line is the last line located by the Line Editor.

### FORMAT:

$$\left[ \text{adr}_1 \left[ \begin{array}{c} \{ ; \} \\ \{ , \} \end{array} \right] \text{adr}_2 \right] \text{S/regexp/string/}$$

or

$$\left[ \text{adr}_1 \left[ \begin{array}{c} \{ ; \} \\ \{ , \} \end{array} \right] \text{adr}_2 \right] \text{!S/regexp/string/} \quad (\text{See Note 3})$$

### ARGUMENTS:

$\text{adr}_1$

Address of the first line to be searched for the specified string of characters. The search begins at the second line in the current buffer. Default: Current line.

$\text{adr}_2$

Address of the last line to be searched for the specified string of characters. Default:  $\text{adr}_1$ .

### NOTE

If both  $\text{adr}_1$  and  $\text{adr}_2$  are omitted, only the current line is searched.

/

(Delimiter) Can be any character that is not in regexp or string. However, the same delimiter must be used in each of the three locations where a delimiter is required.

regexp

String of characters for which the Line Editor is searching; each occurrence of this character string within the specified addresses is replaced with the character(s) specified in the argument "string".



Default: The last regexp specified. This can be determined by entering the ZREGEXP directive, which is described under "Line Editor Debugging Directives" later in this section.

## NOTE

In scanning a line of text for a match of a regular expression, the editor resolves possible ambiguities by selecting the leftmost, shortest possible match first (including a zero length match), with leftmost taking precedence. The substitute directive replaces all such matching substrings of a line in a single left-to-right pass over the line.

string

String of characters that replaces each occurrence of regexp.

## NOTES

1. If the string contains the ampersand (&) character in any position, each occurrence of regexp to be replaced is replaced with regexp included in the string, in place of &. For example, if regexp is "in" and string is "&to", each occurrence of "in" becomes "into". To ignore the special meaning of &, precede it with !C.
2. The occurrence of a line feed in the string expression determines new-line characters; i.e., point in the resulting line at which the line is to be split into two lines.
3. If the directive name !S is used (as illustrated in the second directive format) and the specified substitution fails, no error message is issued and execution of the command file (if any) continues.

Example 1:

```
S/ABGDEF/ABC linefeed DEF/
```

This Substitute directive searches the current line and (1) replaces each occurrence of ABGDEF with ABCDEF and (2) causes the character string to be split between two lines. ABC is on the first line, and DEF is on the second line.

SUBSTITUTE

Example 2:

The contents of the current buffer are:

- (1) E
- (2) NTE
- (3) R
- (4) YOUR

1,3S/linefeed key//

After this Substitute directive is entered, the current buffer contains:

- (1) ENTERYOUR

Example 3:

The contents of the current buffer are:

- (1) XXXXX

S/XX\*/Z/

After this Substitute directive is entered, the current buffer contains:

- (1) ZZZZZ

Example 4:

The contents of the current buffer are:

- (1) XXXXX

S/X\*\$/Z/

After this Substitute directive is entered, the current buffer contains:

- (1) Z

In the following examples, the contents of the current buffer are:

- (1) AAACCC
- (2) BBAAAA (current line)
- (3) CCCBBB
- (4) DDDAAA

Example 5:

2,4S/AAA/XXX/

This Substitute directive searches lines 2 through 4 and replaces each occurrence of AAA with XXX. After this directive is executed, the current buffer contains:

- (1) AAACCC
- (2) BBBXXX
- (3) CCCBBB
- (4) DDDXXX (current line)

Example 6:

.,4S-CCC-UUU-

This Substitute directive searches the current line (line 2) through line 4 and replaces each occurrence of CCC with UUU. After this directive is executed, the current buffer contains:

- (1) AAACCC
- (2) BBAAAA
- (3) UUUBBB
- (4) DDDAAA (current line)

Example 7:

-1,/DDD/S//&JJJ/

This Substitute directive searches one line before the current line (line 1) through the first line that contains DDD (line 4) and replaces each occurrence of DDD with DDDJJJ. After this directive is executed, the current buffer contains:

- (1) AAACCC
- (2) BBAAAA
- (3) CCCBBB
- (4) DDDJJJAAA (current line)

# WRITE

## WRITE (W)

Write a specified line or a series of lines in the current buffer to a specified file. If the file does not already exist, a new file is created with the specified file name. If the named file does exist and currently contains other data, the line(s) written to the file via the Write directive replace the existing contents.

To save the results of previously specified Line Editor directives, you must specify the Write directive before you terminate execution of the Line Editor (i.e., Write must be specified before Quit).

The Write directive must be the last directive on a line. After the Write directive is executed, the specified line(s) remain in the current buffer; a copy of them is written to the specified file.

### FORMAT:

$$\left[ \text{adr}_1 \left[ \left\{ \begin{array}{l} ; \\ ' \end{array} \right\} \text{adr}_2 \right] \right] \text{W}[\text{path}]$$

### ARGUMENTS:

$\text{adr}_1$

Address of the first line to be written to a specified file. Default: First line in the current buffer.

$\text{adr}_2$

Address of the last line to be written to a specified file. Default: Last line in the current buffer.

### NOTE

If both  $\text{adr}_1$  and  $\text{adr}_2$  are omitted, all lines in the current buffer are written to the specified file.

$\text{path}$

Pathname of the file to which the specified line(s) will be written. (Methods of specifying pathnames are described in Section 2.) The pathname may be preceded by any number of spaces. Default: Pathname specified in the latest Read or Write directive associated with the current buffer. If a pathname was not previously specified, an error message is issued.

Example 1:

W IDENT

This Write directive writes all lines in the current buffer to a file whose simple pathname is IDENT.

Example 2:

This example illustrates use of a Write directive in a sample Line Editor session. In this example, there is a file named EXIST that contains the following lines:

- (1) AAA
- (2) BBB
- (3) CCC
- (4) DDD

R EXIST

Read into the current buffer the contents of the file named EXIST. The current buffer contains:

- (1) AAA
- (2) BBB
- (3) CCC
- (4) DDD (current line)

1,\$S/AAA/XXX/

Search each line in the current buffer and change each occurrence of AAA to XXX. The buffer contains:

- (1) XXX
- (2) BBB
- (3) CCC
- (4) DDD (current line)

1,3W

Write lines 1 through 3 to the file specified in the last Read or Write directive; i.e., EXIST. EXIST contains:

- (1) XXX
- (2) BBB
- (3) CCC

Q.

Terminate execution of the Line Editor.

## ADVANCED FUNCTIONS OF THE LINE EDITOR

The directives described on the previous pages permit you to create a source unit and perform basic editing. The following subsections describe Line Editor directives that perform general advanced functions, permit usage of auxiliary buffers, and perform debugging and programming functions. Within each subsection the directives are summarized and then described in detail alphabetically by full directive name.

### GENERAL ADVANCED LINE EDITOR DIRECTIVES

The general advanced Line Editor directives have the following capabilities:

- Cause another specified directive to act on only those lines that do not contain a specified character string (Exclude directive)
- Permit execution of a command instead of Line Editor directives without exiting from the Line Editor (Execute directive)
- Cause another specified directive to act on only those lines that contain a specified character string (Global directive)
- Send line feed to user-out file and error-out file (Line Feed directive)
- Convert the specified expression to lowercase (Lowercase directive)
- Make a different line the current line (New Current Line directive)
- Print the line number of a specified line in the current buffer (Print Line Number directive)
- Print the line number and contents of specified line(s) in the current buffer (Print With Line Number directive)
- Convert the specified expression to uppercase (Uppercase directive).

# EXCLUDE

## EXCLUDE (V)

Exclude specified elements. The Exclude directive can be used in conjunction with Delete, Print, Print Line Number, and Print With Line Number directives so that the specified directive acts on only those lines that do not contain a specified character string.

After the Exclude directive is executed, the current line is the last line searched by the Line Editor.

### FORMAT:

$$\left[ \text{adr}_1 \left[ \begin{array}{c} \{ ; \} \\ \{ , \} \end{array} \right] \text{adr}_2 \right] \text{vx/regexp/}$$

### ARGUMENTS:

$\text{adr}_1$

Address of the first line to be searched. Default: First line in the current buffer.

$\text{adr}_2$

Address of the last line to be searched. Default: Last line in the current buffer.

### NOTE

If both  $\text{adr}_1$  and  $\text{adr}_2$  are omitted, all lines in the buffer are searched.

x

Directive name with which the Exclude directive is being issued; must be one of the following:

D - Delete line(s) that do not contain regexp.

P - Print the contents of line(s) that do not contain regexp.

!P - Print the line number(s) and contents of line(s) that do not contain regexp.

= - Print the line number(s) of line(s) that do not contain regexp.

## EXCLUDE

(Delimiter) Can be any character that does not occur in regexp. The same delimiter must be used before and after regexp.

### regexp

String of characters for which the Line Editor searches; only lines that do not contain regexp are acted upon by the Line Editor during execution of the directive name specified in argument x.

In the following examples, the contents of the current buffer are:

- (1) JJJKKK (current line)
- (2) LLLMMM
- (3) NNNPPP
- (4) RRRJJJ

### Example 1:

```
1,3V!P/JJJ/
```

This Exclude Print with line number directive causes the Line Editor to search lines 1 through 3 and to print the line number and contents of each line that does not contain JJJ.

### Printout:

```
2 LLLMMM
3 NNNPPP
```

Current line: 3

### Example 2:

```
VD*JJJ*
```

This Exclude Delete directive deletes each line that does not contain JJJ; since no addresses are specified, each line in the current buffer is searched.

After this directive is executed, the current buffer contains:

- (1) JJJKKK
- (2) RRRJJJ (current line)



# EXECUTE

## EXECUTE (E)

Cause execution processing. The Execute directive permits you to execute a command instead of Line Editor directives without exiting from the Line Editor; i.e., you can enter any command and then continue to use the Line Editor. For example, the Execute directive can be used to designate a printer as the Line Editor output file. Otherwise, if you want a printout of Line Editor output, the printout is issued to the terminal, which is the original user-out file. If the user-out file is a line printer and a Quit directive is entered to exit from the Line Editor, the user-out file remains set to the printer.

The Execute directive must be the last directive on a line.

The current line is not affected by Execute directives.

### FORMAT:

E command

### ARGUMENT:

command

Any command (see the Commands manual).

### Example:

E FO >SPD>LPT00

This Execute directive includes a File Out (FO) command, which sets the user-out file to the line printer whose pathname is >SPD>LPT00.

# GLOBAL

## GLOBAL (G)

Act on only those lines that contain a specified character string and can be used in conjunction with Delete, Print, Print Line Number, and Print With Line Number directives.

After the Global directive is executed, the current line is the last line searched by the Line Editor.

### FORMAT:

$$\left[ \text{adr}_1 \left[ \begin{array}{l} \{ ; \} \\ \{ , \} \end{array} \right] \text{adr}_2 \right] \text{Gx/regexp/}$$

### ARGUMENTS:

$\text{adr}_1$   
Address of the first line to be searched. Default: First line in the current buffer.

$\text{adr}_2$   
Address of the last line to be searched. Default: Last line in the current buffer.

### NOTE

If both  $\text{adr}_1$  and  $\text{adr}_2$  are omitted, all lines in the current buffer are searched.

x

Directive name with which the Global is being used; must be one of the following:

- D - Delete all line(s) in the specified range containing regexp.
- P - Print the contents of line(s) containing regexp.
- !P - Print the line number(s) and contents of line(s) containing regexp (see "Print With Line Number Directive" later in this section).
- = - Print the line number(s) of line(s) containing regexp (see "Print Line Number Directive" later in this section).

(Delimiter) Can be any character that does not occur in regexp. The same delimiter must be used before and after regexp.

regexp

String of characters for which the Line Editor searches; only lines that contain regexp are acted upon by the directive name specified in argument x.

In the following examples, the contents of the current buffer are:

- (1) JJJKKK
- (2) LLLMMM
- (3) NNNPPP
- (4) RRRJJJ

Example 1:

1,3G!P/JJJ/

This Global Print With Line Number directive causes the Line Editor to search lines 1 through 3 and print the line number and contents of each line that contains JJJ.

Printout:

1 JJJKKK

Current line: 3

Example 2:

GD\*JJJ\*

This Global Delete directive deletes each line that contains JJJ; since no addresses are specified, all lines in the buffer are searched.

After this directive is executed, the current buffer contains:

- (1) LLLMMM
- (2) NNNPPP (current line)

# LINE FEED

## LINE FEED (L OR !L)

Send line feeds to the user-out file and the error-out file, respectively. After the Line Feed directive is executed, the current line is unchanged. Default: none (addresses are ignored).

### FORMAT:

L OR !L

# LOWERCASE

## LOWERCASE (U)

Convert all occurrences of a specified expression within specified addresses from uppercase to lowercase. After the Lowercase directive is executed, the current line is the last line read.

### FORMAT:

$$\left[ \text{adr}_1 \left\{ \begin{array}{l} ; \\ , \end{array} \right\} \text{adr}_2 \right] \text{U/regexp/}$$

### ARGUMENTS:

$\text{adr}_1$

Address of the first line to be searched. Default: Current line.

$\text{adr}_2$

Address of the last line to be searched. Default:  $\text{adr}_1$ .

regexp

String of characters for which the Line Editor searches. Only uppercase letters (A through Z) are converted; others are not changed.

### Example:

U/ADR/

This Lowercase directive searches the current line and changes each occurrence of ADR to adr. If the current line is:

ADR FIRST

after the Lowercase directive is executed, the line contains:

adr FIRST

# NEW CURRENT LINE

## NEW CURRENT LINE (N)

Cause the specified line to become the new current line. The contents of the new current line are not printed after the directive is executed.

### FORMAT:

adrN

### ARGUMENT:

adr

Address of the line that is to be the new current line.

### Example:

/CCC/N

If the following condition exists prior to execution of the N directive:

AAA (current line)  
BBB  
CCC  
DDD

The situation is as follows after the N directive is executed.

AAA  
BBB  
CCC (current line)  
DDD

## PRINT LINE NUMBER

### PRINT LINE NUMBER (= /!P)

Print out the line number of a specified line in the current buffer.

The printout is issued to the user-out file (i.e., the file designated in the -OUT out\_path argument of the Enter Batch Request (EBR) or Enter Group Request (EGR) command) unless that file was reassigned.

After this directive is executed, the current line is the line whose line number was typed.

#### FORMAT:

[adr]=

#### ARGUMENT:

adr

Address of the line whose line number is to be typed.  
Default: Current line.

In the following examples the contents of the current buffer are:

- (1) AAABBB (current line)
- (2) CCCDDD
- (3) CCCEEE

#### Example 1:

/CCC/=

This Print Line Number directive causes a printout of the line number of the first line that contains CCC.

#### Printout:

2

Current line: 2

PRINT LINE NUMBER

Example 2:

=

This Print Line Number directive causes a printout of the line number of the current line.

Printout:

1

Current line: 1



## PRINT WITH LINE NUMBER

### PRINT WITH LINE NUMBER (!P)

Print out the line number and contents of a single line or consecutive lines in the current buffer. The printout is issued to the user-out file, i.e., the file designated in the -OUT out\_path argument of the Enter Batch Request or Enter Group Request command, unless the file was reassigned. If the printout occurs on a terminal, each line of text is preceded by the group identification characters.

After this directive is executed, the current line is the last line whose line number and contents were typed.

#### FORMAT:

$$\left[ \text{adr}_1 \left[ \begin{array}{c} \{ ; \} \\ \{ , \} \end{array} \right] \text{adr}_2 \right] !P$$

#### ARGUMENTS:

adr<sub>1</sub>

Address of the first line whose line number and contents are to be typed. Default: Current line.

adr<sub>2</sub>

Address of the last line whose line number and contents are to be typed. Default: Address specified for adr<sub>1</sub>.

#### NOTE

If both adr<sub>1</sub> and adr<sub>2</sub> are omitted, the line number and contents of the current line print out.

In the following examples, the contents of the current buffer are:

- (1) AAA
- (2) BBB (current line)
- (3) CCC
- (4) DDD

PRINT WITH LINE NUMBER

Example 1:

1,\$!P

This Print With Line Number directive causes a printout of the line number and contents of each line in the current buffer.

Printout:

1 AAA  
2 BBB  
3 CCC  
4 DDD

Current line: 4

Example 2:

!P

This Print With Line Number directive causes a printout of the line number and contents of only the current line.

Printout:

2 BBB

Current line: 2

# UPPERCASE

## UPPERCASE (!U)

Convert all occurrences of a specified expression within specified addresses from lowercase to uppercase.

After the Uppercase directive is executed, the current line is the last line read.

### FORMAT:

$$\left[ \text{adr}_1 \left[ \begin{array}{l} \{ ; \} \\ \{ , \} \end{array} \right] \text{adr}_2 \right] !U/\text{regexp}/$$

### ARGUMENTS:

$\text{adr}_1$

Address of the first line to be searched. Default: Current line.

$\text{adr}_2$

Address of the last line to be searched. Default:  $\text{adr}_1$ .

regexp

String of characters for which the Line Editor searches. Only lowercase letters (a through z) are converted; others are not changed.

### Example:

```
!U/adr/
```

This Uppercase directive searches the current line and changes each occurrence of `adr` to `ADR`. If the current line is:

```
adr first
```

after the Uppercase directive is executed, the line contains:

```
ADR first
```

# COMMENT

## COMMENT (")

Annotate Line Editor command files. The text after the Comment directive appears as program output but is ignored by the Line Editor.

### FORMAT:

"comment

## AUXILIARY BUFFER DIRECTIVES AND ESCAPE SEQUENCES

In the previous pages of this section, it was assumed that there is only a single buffer, the current buffer. The current buffer must be used, but one or more additional buffers, called auxiliary buffers, also can be used. There are 64 auxiliary buffers available for use.

The most common use of auxiliary buffers is for moving or copying text from one part of a file to another.

To make an auxiliary buffer available and to put lines into it, specify the Move, Move-Append, Copy, and/or Copy-Append directives, which are described in the following paragraphs.

Lines cannot be written directly from an auxiliary buffer to a file; the auxiliary buffer must be designated in the Change Buffer directive as the current buffer or the lines must be read back to the current buffer via the escape sequence !P, which is described under "Change Origin of Text During Input Mode," later in this section. Lines can be written from the current buffer to a file via the Write directive (see "Write (W)" earlier in this section).

You can determine the status of each buffer currently in use by specifying the Buffer Status directive.

Auxiliary buffer directives have the following functions:

- Cause Editor to accept a line from terminal (Accept Single Line From a Terminal directive)
- Determine status of each buffer in use (Buffer Status directive)
- Make specified auxiliary buffer the current buffer (Change Buffer directive)
- Cause Line Editor to accept subsequent text from a specified auxiliary buffer
  - During edit mode (Change Origin of Text During Edit Mode directive)
  - During input mode (Change Origin of Text During Input Mode directive)
- Copy line(s) in current buffer to specified auxiliary buffer; lines in current buffer are not deleted.
  - Delete existing lines in auxiliary buffer (Copy directive)
  - Do not delete lines in auxiliary buffer (Copy-Append directive)

- Destroy a buffer (i.e., release its file space) (Destroy directive)
- Move line(s) from current buffer to specified auxiliary buffer; lines in current buffer are deleted
  - Lines overlay existing lines, if any, in auxiliary buffer (Move directive)
  - Lines appended to existing lines, if any, in auxiliary buffer (Move-Append directive).

# ACCEPT SINGLE LINE FROM A TERMINAL

## ACCEPT SINGLE LINE FROM A TERMINAL (!R)

Permit a single line of directives or text to be entered through a terminal. !R normally is used when Line Editor directives are being executed from a buffer. When the Line Editor encounters !R, the entire escape sequence is removed from the input stream and replaced with the line read from the user-in file.

FORMAT:

!R

Example:

```
T/ENTER YOUR NAME/  
A!R!F
```

These directives are in the buffer that is being executed.

The following message appears on the terminal:

```
ENTER YOUR NAME
```

You respond with your name; i.e., Jane Jones.

Following the current line in the current buffer what you entered appears:

```
Jane Jones
```

# BUFFER STATUS

## BUFFER STATUS (X)

Cause a message on the status of each buffer currently in use. The current line is not changed.

### FORMAT:

X

### DESCRIPTION:

The following information is designated:

- Name of each buffer. The original current buffer is always named 0.
- Number of lines in each buffer.
- Indicator as to which buffer is the current buffer. The name of the current buffer is preceded by ->.

If a buffer has been read into and/or written from, the message includes the pathname specified in the last read or write.

If the contents of the current buffer have been modified (i.e., in the message, MOD is designated before its name), all of the following conditions must exist:

- Lines from an existing file have been read into the current buffer via a Read directive or the contents of the current buffer have been written to a file.
- The contents of the buffer were modified via one or more Line Editor directives.

Each message has the following format:

|                  |    |       |               |             |
|------------------|----|-------|---------------|-------------|
| number of lines  | -> | [MOD] | (buffer-name) | [pathname]  |
| [number of lines |    | [MOD] | (buffer-name) | [pathname]] |
| :                |    | :     | :             | :           |
| :                |    | :     | :             | :           |
| :                |    | :     | :             | :           |



Example:

This example illustrates usage of the Buffer Status directive. The file USE, which is in the working directory, comprises the following lines:

- (1) AAA (current line)
- (2) BBB
- (3) CCC
- (4) DDD

R USE

Read the contents of USE into the current buffer, which is named 0.

1,\$S\*BBB\*XXX\*

Search the first line through the last line in the current buffer and change each occurrence of BBB to XXX. After this directive is executed, the current buffer contains:

- (1) AAA
- (2) XXX
- (3) CCC
- (4) DDD

3,4M2

Move lines 3 and 4 of the current buffer into auxiliary buffer 2. After this directive is executed, the current buffer contains:

- (1) AAA
- (2) XXX

Auxiliary buffer 2 contains:

- (1) CCC
- (2) DDD

X

Request the status of each buffer currently in use. The following message is issued:

```
2 ->MOD (0) USE
2      (2)
```

# CHANGE BUFFER

## CHANGE BUFFER (Bx)

Designate that a specified auxiliary buffer is to become the current buffer. The previously designated current buffer becomes an auxiliary buffer.

After this directive is executed, lines can be written from the new current buffer to a file.

### FORMAT:

Bx

### ARGUMENT:

x

Buffer name. The name must be 1 to 6 ASCII characters. If the name comprises more than a single character, the name must be enclosed within parentheses; otherwise, the parentheses are optional. The original current buffer name is 0. This name can never be altered. An auxiliary buffer name, once specified, cannot be altered during the current Line Editor session.

### Example:

B3

This directive designates that auxiliary buffer 3 is the current buffer. If desired, lines can now be written from this buffer to a file.

## CHANGE ORIGIN OF TEXT DURING EDIT MODE

### CHANGE ORIGIN OF TEXT DURING EDIT MODE (!B)

Cause the Line Editor to read subsequent directives from a specified auxiliary buffer. !B can be specified within an expression, pathname, text to be typed (i.e., in the Type directive), or as a directive. When the Line Editor encounters this sequence in an expression, pathname, or text, the entire escape sequence is removed from the input stream and replaced with the literal contents of the first line of the specified buffer; if !B is a directive, the input stream is replaced with the entire literal contents of the specified buffer. If another !B escape sequence is encountered while accepting input from buffer x, the newly encountered escape sequence is also replaced by the contents of its named buffer.

The buffer to which the input stream is redirected may contain Line Editor requests, literal text, or both. If the Line Editor is executing a request obtained from an auxiliary buffer and an error occurs, the usual error comment is suppressed and the remaining contents of that buffer are skipped. Control returns to the statement immediately following the !B escape sequence that called the auxiliary buffer. For example, if one thinks of the escape sequence !B(x) as a subroutine call statement, the failure to match a regular expression specified by some request in buffer x may be thought of as a return statement. Once the last commands in the auxiliary buffer have been processed, control returns to the statement immediately following the !B escape sequence that called the auxiliary buffer.

The buffer name can be in the format (ARGn), where n is a number from 1 to 9 that refers to the nth argument that followed the -ARG argument of the ED command. The escape sequence is replaced with the first (or only) line of the buffer (ARGn) created during initialization of the Line Editor.

#### FORMAT:

!Bx

#### ARGUMENT:

x

Name of the buffer that contains subsequent Line Editor text. The buffer name must be 1 through 6 ASCII characters. If the buffer name comprises more than a single character, the name must be enclosed within parentheses; otherwise, the parentheses are optional.

CHANGE ORIGIN OF TEXT  
DURING EDIT MODE

Example 1: !B as a directive

!B(TEST)

In this example, the contents of the current buffer and the auxiliary buffer named TEST are:

Current buffer:

- (1) A
- (2) B
- (3) A
- (4) D
- (5) E

Auxiliary buffer:

1,\$S/A/X/

This Substitute directive designates that in the current buffer all occurrences of A be replaced with X. After the Substitute directive is executed, the current buffer contains:

- (1) X
- (2) B
- (3) X
- (4) D
- (5) E

The auxiliary buffer named TEST remains the same.

Example 2: !B Within an Expression

2S/AAA/!B2/

This Substitute directive designates that in the second line of the current buffer, each occurrence of AAA should be replaced with the first line of auxiliary buffer 2.

The contents of the current buffer and auxiliary buffer 2 are:

Current buffer:

- (1) AAABBB
- (2) CCCAAA
- (3) XXXYYY

Auxiliary buffer 2:

DDD  
EEE

After the Substitute directive is executed, the current buffer contains:

(1) AAABBB  
(2) CCCDDD  
(3) XXXYYY

Example 3: !B Within Text to be Typed

T/!B2/

This Type directive (which is described later in this section) requests that the first line of auxiliary buffer B2 be displayed on the user-out file.

Example 4: Buffer Name (ARGn)

The ED command includes the argument -ARG ABC "MY NAME" XYZ

S/DEF/!B(ARG3)/

This Substitute directive searches the current line and replaces each occurrence of DEF with XYZ (i.e., the third argument following -ARG in the ED command).

# CHANGE ORIGIN OF TEXT DURING INPUT MODE

## CHANGE ORIGIN OF TEXT DURING INPUT MODE (!B)

Cause the Line Editor to accept subsequent text from a specified auxiliary buffer. The escape sequence !B can appear within the text of an Input directive.

When the Line Editor encounters !B, the entire escape sequence is removed from the input stream and replaced with the literal contents of the specified buffer. If another !B escape sequence is encountered after accepting text from the specified buffer, the newly encountered escape sequence is also replaced with the contents of the named buffer.

### FORMAT:

```
[text]!Bx [ [text]!B ] ...
```

### ARGUMENT:

x

Name of the buffer that contains subsequent Line Editor text. The buffer name must be 1 to 6 ASCII characters. If the buffer name comprises more than a single character, the name must be enclosed within parentheses; otherwise, the parentheses are optional.

### Example

```
/D/I  
!B(TEST)!F
```

In this example, the contents of the current buffer and the auxiliary buffer named TEST are:

### Auxiliary buffer:

```
(1) X  
(2) Y  
(3) Z
```

### Current buffer:

```
(1) A  
(2) B  
(3) C  
(4) D  
(5) E
```

CHANGE ORIGIN OF TEXT  
DURING INPUT MODE

This Insert directive designates that the contents of the auxiliary buffer named TEST to be inserted into the current buffer before the line that contains D.

After the Insert directive is executed, the current buffer contains:

- (1) A
- (2) B
- (3) C
- (4) X
- (5) Y
- (6) Z
- (7) D
- (8) E

The auxiliary buffer named TEST remains the same.

# COPY

## COPY (K)

Write into a specified auxiliary buffer a single line or consecutive lines contained in the current buffer. The lines in the current buffer are not deleted; i.e., the lines are in both the current and the auxiliary buffers. Any lines previously in the auxiliary buffer are destroyed during execution of the Copy directive.

After the Copy directive is executed, the current line in the current buffer is the last line moved to the auxiliary buffer. There is no current line in the auxiliary buffer until that auxiliary buffer is changed to the current buffer via a Change Buffer directive.

### FORMAT:

$$\left[ \text{adr}_1 \left[ \left\{ \begin{array}{l} ; \\ ' \end{array} \right\} \text{adr}_2 \right] \right] \text{Kx}$$

### ARGUMENTS:

$\text{adr}_1$

Address of the first line to be written into the specified auxiliary buffer. Default: Current line.

$\text{adr}_2$

Address of the last line to be written into the specified auxiliary buffer. Default:  $\text{adr}_1$ .

### NOTE

If both  $\text{adr}_1$  and  $\text{adr}_2$  are omitted, only the current line is written into the specified auxiliary buffer.

x

Name of the auxiliary buffer into which the specified line(s) are written. The name must be 1 to 6 ASCII characters. If the name comprises more than a single character, the name must be enclosed within parentheses; otherwise, the parentheses are optional.



Example:

1,3K(52)

This Copy directive copies lines 1 through 3 of the current buffer into auxiliary buffer 52. The contents of the current buffer are:

- (1) FIRST (current line)
- (2) SECOND
- (3) THIRD
- (4) FOURTH

After the Copy directive is executed, the contents of the current buffer are unchanged, but the current line is line 4. Auxiliary buffer 52 contains:

- (1) FIRST
- (2) SECOND
- (3) THIRD

There is no current line in the auxiliary buffer.

# COPY-APPEND

## COPY-APPEND (!K)

Write a line or lines from the current buffer to an auxiliary buffer without destroying the contents of the auxiliary buffer. The lines copied from the current buffer are appended to the contents of the auxiliary buffer. The lines written are also retained in the current buffer.

After the Copy-Append directive is executed, the current line in the current buffer is the last line written to the auxiliary buffer. There is no current line in the auxiliary buffer.

### FORMAT:

$$\left[ \text{adr}_1 \left[ \left\{ \begin{array}{l} ; \\ ' \end{array} \right\} \text{adr}_2 \right] \right] !Kx$$

### ARGUMENTS:

$\text{adr}_1$

Address of the first line to be written to the specified auxiliary buffer. Default: Current line.

$\text{adr}_2$

Address of the last line to be written to the specified auxiliary buffer. Default:  $\text{adr}_1$ .

### NOTE

If both addresses are omitted, only the current line is written to the auxiliary buffer.

x

Name of the auxiliary buffer into which the specified line(s) are written. The name must be from 1 to 6 ASCII characters. If the name is more than one character, it must be enclosed within parentheses; otherwise, parentheses are optional.

Example:

1,3!K(ABUF)

This directive appends lines 1 through 3 of the current buffer to the contents of auxiliary buffer ABUF. Thus, if the current buffer and ABUF contain the following lines prior to execution:

| <u>Current</u>         | <u>ABUF</u> |
|------------------------|-------------|
| (1) AAA (current line) | (1) MMM     |
| (2) BBB                | (2) NNN     |
| (3) CCC                |             |
| (4) DDD                |             |

They will contain the following after execution:

| <u>Current</u>         | <u>ABUF</u> |
|------------------------|-------------|
| (1) AAA                | (1) MMM     |
| (2) BBB                | (2) NNN     |
| (3) CCC (current line) | (3) AAA     |
| (4) DDD                | (4) BBB     |
|                        | (5) CCC     |

# DESTROY

## DESTROY (^B)

Release a specified auxiliary buffer's file space. Any buffer other than buffer 0 and the current buffer can be removed; if the current buffer name is specified, the directive is ignored and an error message is issued.

### FORMAT:

^Bx

### ARGUMENT:

x

Name of the auxiliary buffer to be destroyed. The name must be from 1 to 6 ASCII characters. If the name comprises more than one character, it must be enclosed within parentheses; otherwise, parentheses are optional.

### Example:

^B(AX)

This Destroy directive removes buffer AX.

# MOVE

## MOVE (M)

Move a single line or consecutive lines from the current buffer to a specified auxiliary buffer; the lines no longer exist in the current buffer. If the auxiliary buffer already contains lines, those lines are destroyed.

After the Move directive is executed, the current line in the current buffer is the next line (if any) after the last line moved to the auxiliary buffer. There is no current line in the auxiliary buffer, nor in the current buffer if all its lines were moved.

### FORMAT:

$$\left[ \text{adr}_1 \left[ \left\{ \begin{array}{l} ; \\ , \end{array} \right\} \text{adr}_2 \right] \right] \text{Mx}$$

### ARGUMENTS:

$\text{adr}_1$

Address of the first line to be moved from current buffer to auxiliary buffer. Default: Current line.

$\text{adr}_2$

Address of the last line to be moved from current buffer to auxiliary buffer. Default:  $\text{adr}_1$ .

### NOTE

If both  $\text{adr}_1$  and  $\text{adr}_2$  are omitted, only the current line is moved from the current buffer to the auxiliary buffer.

x

Name of the auxiliary buffer to which the specified line(s) are moved. The name must be 1 to 6 ASCII characters. If the name comprises more than a single character, the name must be enclosed within parentheses; otherwise, the parentheses are optional.

MOVE

Example:

1,3M5

This Move directive moves 1 through 3 from the current buffer to auxiliary buffer 5. In this example, the contents of the current buffer are:

- (1) FIRST (current line)
- (2) SECOND
- (3) THIRD
- (4) FOURTH

After the Move directive is executed, the current buffer contains:

- (1) FOURTH (current line)

Auxiliary buffer 5 contains:

- (1) FIRST
- (2) SECOND
- (3) THIRD

# MOVE-APPEND

## MOVE-APPEND (!M)

Move one or more lines of text from the current buffer to the specified auxiliary buffer. The lines are appended to the existing contents of the auxiliary buffer; the existing contents of the auxiliary buffer are not overlaid. If the auxiliary buffer contains no text, the lines are placed in the auxiliary buffer starting at line 1. The lines moved are deleted from the current buffer.

After the Move-Append directive is executed, the current line in the current buffer is the next line (if any) after the last line written to the auxiliary buffer. There is no current line in the auxiliary buffer, nor in the current buffer if all lines there were written out.

### FORMAT:

$$\left[ \text{adr}_1 \left[ \begin{array}{c} \{ ; \} \\ \{ , \} \end{array} \right] \text{adr}_2 \right] !Mx$$

### ARGUMENTS:

$\text{adr}_1$

Address of the first line to be moved from the current buffer to the auxiliary buffer. Default: Current line.

$\text{adr}_2$

Address of the last line to be moved from the current buffer to the auxiliary buffer. Default:  $\text{adr}_1$ .

### NOTE

If both  $\text{adr}_1$  and  $\text{adr}_2$  are omitted, only the current line is moved from the current buffer to the auxiliary buffer.

x

Name of the auxiliary buffer to which the specified line(s) are moved. The name must be 1 to 6 ASCII characters. A name of more than one character must be enclosed in parentheses; otherwise, parentheses are optional.

MOVE-APPEND

Example:

1,3!M(SOOZ)

This directive appends lines 1 through 3 to the contents of auxiliary buffer SOOZ. If the contents of the buffers are as follows prior to the move:

| <u>Current</u>           | <u>SOOZ</u> |
|--------------------------|-------------|
| (1) FIRST (current line) | (1) AAAAA   |
| (2) SECOND               | (2) BBBB    |
| (3) THIRD                |             |
| (4) FOURTH               |             |

The buffers contain the following after the move:

| <u>Current</u>            | <u>SOOZ</u> |
|---------------------------|-------------|
| (1) FOURTH (current line) | (1) AAAAA   |
|                           | (2) BBBB    |
|                           | (3) FIRST   |
|                           | (4) SECOND  |
|                           | (5) THIRD   |



## LINE EDITOR DEBUGGING DIRECTIVES

The functions of Line Editor debugging directives are:

- Print contents of specified line(s) on the terminal (Hexadecimal Dump directive)
- Display, on the user-out file, the last specified regular expression (ZREGEXP directive)
- Display each directive line before it is executed (ZTRACE directive).

# HEXADECIMAL DUMP

## HEXADECIMAL DUMP (ZDUMP)

Print the contents of specified line(s) on the terminal in both hexadecimal and ASCII formats. The output format consists of the line number, the length (number of characters) expressed in hexadecimal, eight words in hexadecimal format, and eight words in ASCII format.

The display of each buffer line is separated from following displays by a blank line. If a buffer line is too long to be displayed on a single line, it is continued on the next line, with no blank line separation.

After this directive is executed, the current line is the last (or only) line printed.

FORMAT:

$$\left[ \text{adr}_1 \left[ \begin{array}{l} \{ ; \} \\ \{ , \} \end{array} \right] \text{adr}_2 \right] \text{ZDUMP}$$

ARGUMENTS:

$\text{adr}_1$

Address of the first buffer line to be dumped. Default: Current line.

$\text{adr}_2$

Address of the last buffer line to be dumped. Default:  $\text{adr}_1$ .

### NOTE

If both addresses are omitted, only the current line is dumped.

Example:

The contents of lines 1 and 2 of the current buffer are:

- (1) START EDIT
- (2) VDEF ZFVER,X'3031'

1,2ZDUMP

# HEXADECIMAL DUMP

This Hexadecimal Dump directive produces the following output at the terminal:

```
0001 000A 5354 4152 5420 4544 4954          START EDIT
0002 0012 5644 4546 205A 4656 4552 2C58 2733 3033 VDEF ZFVER,X'303
      3727                                     1'
```

Thus, 0001 indicates line 1; 000A indicates a length of 10 characters (A ), followed by the hexadecimal equivalent of START EDIT. A blank line is followed by the dump of line 2, with a length of 18 characters (12 ). Because nine words are required to fully dump the line, the output continues on the next line of the terminal, with no blank line intervening.

# ZREGEXP

## ZREGEXP

Display the last specified expression on the user-out file.  
The current line is not changed.

FORMAT:

ZREGEXP

Example:

S/ABC/DEF/  
ZREGEXP

This ZREGEXP directive displays the last specified expression;  
i.e., /ABC/.

# ZTRACE

## ZTRACE

Display each directive line on the user-out file before it is executed.

### FORMAT:

ZTRACE { ON }  
          { OFF }

### ARGUMENTS:

- ON Each directive line is displayed before it is executed.
- OFF Subsequent lines are not displayed before they are executed.

### Example:

This example illustrates a program that includes an ED command to load the Line Editor and a ZTRACE ON directive. Following is a printout of the Line Editor output.

Program including ED command and ZTRACE ON directive:

```
1  RL DIRECTORY
2  FO DIRECTORY
3  WS &l "LS -BF"
4  FO
5  &A
6  ED
7  ZTRACE ON
8  B1
9  I
10 R DIRECTORY
11 GD/^ &/
12 GD/^ . ENTRY NAME TYPE$/
13 GD/ D$/
14 l,$S/^ . //
15 l,$S/^DIRECTORY: . //
16 $N
17 :C ?/^/^/;M(2)
18 :D ^*/^*/S/^.*$/& !C!B2>&/?+1;>D
19 ?+1,-1N>C
20 */^*/D!F
21 B0
22 !B1
23 W DIRECTORY
24 Q
```

```

25 ED -NBS -LL 160
26 R DIRECTORY
27 1,$S/..$/
28 1,$S^.....
29 ,$$S/!.*$//
30 1,$S/^/!H00/
31 W DIRECTORY
32 Q
33 SORT -IN SORT_CMD_ST -FF
34 F0 >SPD>LPT00
35 PR SORTED_DIR -LL 132
36 F0

```

Line Editor output:

```

EDIT-0200-09/11/0948
**EDIT** B1
**EDIT** I
**INPUT** R DIRECTORY
**INPUT** GD/^ $/
**INPUT** GD/^ . ENTRY NAME TYPE$/
**INPUT** GD/ D$/
**INPUT** 1,$S/^ . //
**INPUT** 1,$S/^ DIRECTORY: . //
**INPUT** $N
**INPUT** :C ?/^~/;M(2)
**INPUT** :D ^*/^~/S/^.*S/&!B2>&/?+1;>D
**INPUT** ?+1,-1N>C
**INPUT** */^~/D!F
**EDIT** B0
**EDIT** !B1
**EDIT** R DIRECTORY
**EDIT** GD/^ $/
**EDIT** GD/^ . ENTRY NAME TYPE$/
**EDIT** GD/ D$/
**EDIT** 1,$S/^ . //
**EDIT** 1,$S/^ DIRECTORY: . //
**EDIT** $N
**EDIT** :C ?/^~/;M(2)
**EDIT** :D ^*/^~/S/^.*S/&!B2>&/?+1;>D
**EDIT** :D ^*/^~/S/^.*S/&!B2>&/?+1;>D
**EDIT** :D ^*/^~/S/^.*S/&!B2>&/?+1;>D
**EDIT** :D ^*/^~/S/^.*S/&!B2>&/?+1;>D
**EDIT** ?+1,-1N>C
**EDIT** :C ?/^~/;M(2)
**EDIT** :D ^*/^~/S/^.*S/&!B2>&/?+1;>D
**EDIT** :D ^*/^~/S/^.*S/&!B2>&/?+1;>D
**EDIT** :D ^*/^~/S/^.*S/&!B2>&/?+1;>D
**EDIT** ?+1,-1N>C
**EDIT** :C ?/^~/;M(2)
**EDIT** :D ^*/^~/S/^.*S/&!B2>&/?+1;>D
**EDIT** :D ^*/^~/S/^.*S/&!B2>&/?+1;>D

```

```
**EDIT** :D ^*/^^/S/^. *S/&!B2>&/?+1;>D  
**EDIT** ?+1,-1N>C  
**EDIT** */^^/D  
**EDIT** W DIRECTORY  
**EDIT** Q
```

## LINE EDITOR PROGRAMMING DIRECTIVES

Line Editor programming directives cause conditional execution of subsequent directives, change the location of subsequent Line Editor input, and display a line of text on the user-out file. Programming directives can be in the directive input file (specified in the -IN path argument of the ED command) or an auxiliary buffer, or they can be entered through a terminal.

Each conditional directive includes one or more other Line Editor directives. The directives must be on a single line. If the specified condition exists, the subsequent embedded directive(s) are executed. The following conditions can be tested:

- Does specified line exist (Address Prefix directive)
- Does current buffer contain data (If Empty and If Data directives)
- Is current line a specified line (If Line and If Not Line directives)
- Is current line within specified lines (If Range and If Not Range directives)
- Is specified expression within specified lines (Search and Search Not directives).

Programming directives also have the following capabilities:

- Change location from which Line Editor accepts subsequent directives (Go To directive)
- Define location that can be the endpoint of a Go To directive (Label directive)
- Display a line of text on the user-out file (Type directive).

### NOTE

If a directive format comprises multiple directives, the directives can be separated by spaces for readability.



# ADDRESS PREFIX

## ADDRESS PREFIX (?)

Execute the directives contained in the Address Prefix line if the specified line exists in the current buffer; otherwise, they are not executed.

### FORMAT:

```
?adr {;} directive [directive] ...
```

### ARGUMENTS:

adr

Address of the line for which the Line Editor searches.

### NOTE

If adr is immediately followed by a semicolon, adr becomes the current line. If adr is immediately followed by a comma, the current line is not changed.

### directive

Any Line Editor directive(s); they are executed only if the specified line is found.

### Example 1:

```
?8;P
```

This Address Prefix directive specifies that if there is a line 8 in the current buffer, print the contents of that line; that line becomes the current line.

### Example 2:

In this example, the contents of the current buffer are:

- (1) DEFGHI
- (2) ABCZYZ
- (3) ABCGGG (current line)

```
?/ABC/;S/ABC/DEF/
```

This Address Prefix directive designates that if there is a line that contains ABC, make that line the current line, and in that line replace each occurrence of ABC with DEF.

ADDRESS PREFIX

After this directive is executed, the current buffer contains:

- (1) DEFGHI
- (2) DEFXYZ (current line)
- (3) ABCGGG

# GO TO

## GO TO (>)

Change the location from which the Line Editor accepts subsequent directives.

If the Go To directive is encountered in the buffer that is currently being executed, the Line Editor accepts subsequent directives from a specified location in that buffer. The location must have been previously defined in that buffer by a label directive.

If the Go To directive is entered interactively, only directives in the current directive line are used.

### FORMAT:

>label

### ARGUMENT:

label

Location to which control is transferred; the Line Editor accepts subsequent directives from this location.

If the label comprises multiple characters, they must be enclosed within parentheses; otherwise, the parentheses are optional.

### Example 1:

In this example, the contents of the current buffer are:

- (1) EAST ROCKAWAY, NY
- (2) LONG BEACH, NY
- (3) BRIGHTON, MASS
- (4) ANDOVER, MASS
- (5) HEWLETT, NY

Buffer 2 contains the following directives:

```
:(REPEAT)1,$P
```

Assign label REPEAT to Print directive line.

```
1,$S/MASS$/MASSACHUSETTS/P
```

Substitute each occurrence of MASS at the end of a line with MASSACHUSETTS and print the contents of the last line in the buffer (i.e., line 5).

GO TO

NOTE

When the Line Editor searches the buffer the second time and does not find MASS at the end of a line, control returns to the previous buffer or to the terminal.

1,\$S/NY/NEW YORK/>(REPEAT)

Substitute each occurrence of NY with NEW YORK and print the contents of all lines (i.e., lines 1 through 5).

Example 2:

:A?/ABC/;S/ABC/DEF/P>A

If this directive is entered interactively, the following actions take place. The information to the right of each action indicates how the action is requested in the directive line.

Assign label A to directive line. :A

If ABC exists, take the subsequent actions. ?/ABC/

Change the current line to the location of ABC ;  
(semicolon precedes the substitute directive).

Replace each occurrence of ABC with DEF. S/ABC/DEF/

Print the current line. P

Go to line A (i.e., reexecute the same directive line). >A

After all lines containing ABC have been acted upon (i.e., each occurrence of ABC has been replaced with DEF and the resulting lines printed), control returns to the next directive entered interactively.

## IF DATA

### IF DATA (#)

Execute the directives contained on the If Data directive line if the current buffer contains data; otherwise, they are not executed.

#### FORMAT:

#directive [directive] ...

#### ARGUMENT:

directive

Any Line Editor directive(s); they are executed only if the current buffer contains data.

## IF EMPTY

### IF EMPTY (^#)

Execute the directives contained in the If Empty directive line if the current buffer is empty; otherwise, they are not executed.

#### FORMAT:

^#directive [directive] ...

#### ARGUMENT:

directive

Any Line Editor directive(s); they are executed only if the current buffer does not contain data.

## IF LINE

### IF LINE (adr#)

Execute the directives contained on the If Line directive line if the current line is the specified line; otherwise, they are not executed.

#### FORMAT:

adr#directive [directive] ...

#### ARGUMENTS:

adr

Address of the line being checked to see if it is the current line.

directive

Any Line Editor directive(s); they are executed only if the specified line is the current line.

# IF NOT LINE

## IF NOT LINE (adr ^#)

Execute the directives on the If Not Line directive line if the current line is not the specified line; otherwise, they are not executed.

### FORMAT:

adr^#directive [directive] ...

### ARGUMENTS:

adr

Address of the line being checked to see if it is the current line.

directive

Any Line Editor directive(s); they are executed only if the specified line is not the current line.



## IF RANGE

### IF RANGE (adr(s) #)

Execute the directives on the If Range directive line if the current line is within specified lines; otherwise, they are not executed.

#### FORMAT:

adr<sub>1</sub> {;} adr<sub>2</sub> #directive [directive] ...  
          {,}

#### ARGUMENTS:

adr<sub>1</sub>

Address of the first line to be searched.

adr<sub>2</sub>

Address of the last line to be searched.

directive

Any Line Editor directive(s); they are executed only if the current line is within addresses adr<sub>1</sub> through adr<sub>2</sub>. The current line is unchanged.

# IF NOT RANGE

## IF NOT RANGE (adrs ^#)

Execute the directives on the If Not Range directive line if the current line is not within specified lines; otherwise, they are not executed.

### FORMAT:

adr<sub>1</sub> {;} adr<sub>2</sub> ^#directive [directive] ...  
          {,}

### ARGUMENTS:

adr<sub>1</sub>

Address of the first line to be searched.

adr<sub>2</sub>

Address of the last line to be searched.

directive

Any Line Editor directive(s); they are executed only if the current line is not within addresses adr<sub>1</sub> through adr<sub>2</sub>. The current line is unchanged.

### Example:

1,10^#S/yes/no/

This If Not Range directive specifies that if the current line is not within lines 1 through 10, substitute each occurrence of "yes" in the current line with "no".

## LABEL

### LABEL (:)

Define a location to which the Line Editor can be directed (via a Go To directive) for subsequent directives. If a Go To directive is entered interactively, only the current directive line is searched for the label. The Label directive must be specified at the beginning of a line.

#### FORMAT:

:labeldirective [directive] ...

#### ARGUMENTS:

label

Location that can be the argument value of a Go To statement; i.e., a location to which control can be transferred. If multiple characters constitute the label, they must be enclosed within parentheses; otherwise, parentheses are optional.

directive

Any Line Editor directive(s); they are executed when control passes to the specified label.

# SEARCH

## SEARCH (\*)

Execute the directives on the Search directive line if a specified expression is within specified lines; otherwise, they are not executed.

### FORMAT:

adr<sub>1</sub> {; } adr<sub>2</sub> \*/regexp/directive [directive] ...

### ARGUMENTS:

adr<sub>1</sub>

Address of the first line to be searched for the regular expression. Default: Current line.

adr<sub>2</sub>

Address of the last line to be searched for the regular expression. Default: adr<sub>1</sub>.

### NOTE

If both adr<sub>1</sub> and adr<sub>2</sub> are omitted, only the current line is searched.

regexp

String of characters for which the Line Editor is searching.

directive

Any Line Editor directive(s); they are executed only if the specified expression is within the specified addresses.

## SEARCH NOT

### SEARCH NOT (^\*)

Execute the directives on the Search Not directive line if a specified expression is not within specified lines; otherwise, they are not executed. The current line is unchanged.

#### FORMAT:

$adr_1 \left\{ \begin{array}{l} ; \\ , \end{array} \right\} adr_2 \text{^*/regexp/directive [directive] ...}$

#### ARGUMENTS:

$adr_1$

Address of the first line to be searched for the regular expression. Default: Current line.

$adr_2$

Address of the last line to be searched for the regular expression. Default:  $adr_1$ .

#### NOTE

If both  $adr_1$  and  $adr_2$  are omitted, the directives are executed only if the regular expression is not in the current line.

regexp

String of characters for which the Line Editor is searching.

directive

Any Line Editor directive(s); they are executed only if the specified expression is not within the specified addresses. The current line is unchanged.

# TYPE

## TYPE (T)

Display a line of text on the user-out file. If the optional exclamation point (!) is specified in the directive format, the next input or output appears immediately after the printout, on the same line; otherwise, the next printouts are on subsequent lines.

### FORMAT:

[!]T/text/

### ARGUMENTS:

/

(Delimiter) Can be any nonblank character, but the same character must be used in each place where a delimiter is required.

text

Text to be displayed. Default: One blank line.

### Example 1:

T/IDENTIFICATION NUMBER/

This Type directive prints IDENTIFICATION NUMBER. Since the optional exclamation point was not specified, subsequent input or output appears on subsequent lines.

### Example 2:

!T/IDENTIFICATION NUMBER !B2/

This Type directive prints IDENTIFICATION NUMBER and the contents of auxiliary buffer B2. If B2 contains FOR THIS YEAR, the printout will be: IDENTIFICATION NUMBER FOR THIS YEAR. Since the directive name T was immediately preceded by an exclamation point, the next input or output appears immediately after the printout, on the same line.

## PROGRAMMING CONSIDERATIONS

1. Tabbing causes embedded tab characters to be replaced with the appropriate number of spaces so that printed output on a printer or terminal has "tab stops" at character position 11 and at every subsequent 10 character positions. Tab characters can be entered into Assembly language source lines by pressing CTRL I on the terminal device while entering insert and/or substitute directive(s). CTRL I is a nonprinting tab character that has a hexadecimal value of 09. Tabbing is not apparent until a printout occurs.
2. The Line Editor uses a minimum of two temporary work files in the working directory. These files are created by the Line Editor when the Line Editor is invoked; they exist only during the current execution of the Line Editor. A minimum of 16 diskette or 8 cartridge sectors must be available in the working directory for temporary work files. Additional temporary files are created for each auxiliary buffer used; the number of temporary files is limited by the space available in the working directory.
3. If you specify a buffer name comprising more than a single character and omit the parentheses, only the first character is considered the buffer name; subsequent characters are treated as directives.
4. If a file manager error (190223, lack of space) or a physical error (190107) is encountered, use the Quit directive to exit from the Line Editor, and restart after the problem has been corrected. Attempting to recover by other means (such as the escape sequence) can cause unspecified results. If an error occurs while processing a work file (this situation is indicated by an error message that is not followed by a file name), the Line Editor can terminate processing, and a fatal error message is issued.
5. An error occurs if the maximum number of lines that the Line Editor accepts in a program has been reached. Control is returned to command level.

## LINE EDITOR PROCEDURES

This section provides information on using the Line Editor to create and modify files. When using the Line Editor, each directive or line of information entered must be followed by a carriage return. Throughout this text, all user entries shown in examples are shaded. This distinguishes user entries from system messages and prompts.

## Initiating A Line Editor Session

To initiate a Line Editor session, enter the ED command followed by a space and a -PT as shown:

```
RDY:  
ED -PT  
Edit -REL -09/09/81  
E?
```

The entry ED -PT causes display of an E? prompt. The E? prompt is caused by the -PT argument and informs you that the Line Editor is ready to accept directives. The E? prompt also indicates that the Line Editor is ready for use.

### CREATING WORK FILES

In addition to the -PT argument, other arguments can be included with the ED command. One argument is the -SF argument. There may be times when you are working on file contents in a temporary work area known as the current buffer. If the system fails while working in the current buffer, the contents of the current buffer are lost. The -SF option creates two permanent work files. If the system fails, current buffer contents are not lost.

The two files created by the -SF argument are the .EDWK1 and .EDWK2 files. The format for the -SF argument is -SF, a space, and a name for the work files. The .EDWK1 and .EDWK2 suffixes are appended to the specified file name automatically. The file name can be from one to six characters long.

The following example uses the -PT and -SF arguments. Immediately following the -SF argument is a space and filename CSRC. This entry creates two permanent files named CSRC.EDWK1 and CSRC.EDWK2 in your current working directory. As modifications are made to the current file, the permanent work files are updated alternately. If the system fails while you are working with the Line Editor, the files named CSRC.EDWK1 and CSRC.EDWK2 are saved.

```
ED -PT -SF CSRC  
Edit REL -09/09/81
```

Once the system is working again, you can sign on, invoke the Line Editor, and read in CSRC.EDWK1 or CSRC.EDWK2 to begin modifying the file again. Reading a file into the current buffer and working on the file are described later.



Two facts about the use of the -SF argument are very important:

- The permanent files with the .EDWK1 and .EDWK2 suffixes are updated alternately. You should check both files to determine which file has the latest version of the update file. Use the X directive (described under "Buffer Status") to determine file status. These files can be read in by the Line Editor only if there is a system failure. When you create the files with the -SF argument, you cannot examine those files. Also, if there is no system failure, the two files are released when you quit the Line Editor. The files are available for examination only after there is a system failure during use of the Line Editor.
- When you sign on and invoke the Line Editor after a system failure, the -SF argument should be followed by a file name that is different from the first backup file name; otherwise, the Line Editor overwrites the old safe files.

Example:

Assume that you have signed on and invoked the Line Editor as shown above. Later, during the building or modification of a file through the current buffer, there is a system failure. Once the system is running again, sign on and invoke the Line Editor again. This time the -SF argument is followed by the file name CTEMP. As a result of this series of terminal entries, the permanent work files named CSRC.EDWK1 and CSRC.EDWK2 are available for examination and a new set of backup files named CTEMP.EDWK1 and CTEMP.EDWK2 are created. In this way, old files can be read into the current buffer for modification and two backup files are available in case there is another system failure. When the new update session is complete, you should release the files named CSRC.EDWK1 and CSRC.EDWK2. If there is no system failure, the files named CTEMP.EDWK1 and CTEMP.EDWK2 are released automatically.

```
RDY:
ED -PT -SF CTEMP
Edit REL -09/09/81
E?
```

#### LINE EDITOR MODES

The Line Editor works in two modes: input mode and edit mode. Input mode is used for adding lines to an existing file or for building a new file. Edit mode is used for making changes to an existing file. In edit mode, deletion of lines, substitution, and printing of lines can be done.

#### QUITTING THE LINE EDITOR

After invoking the Line Editor and finishing your editing session, you will want to quit, or exit, the Line Editor.

To exit the Line Editor, you must be in edit mode. The Line Editor prompt indicates the current mode. If the Line Editor prompt E? is displayed, the Line Editor is in edit mode.

If the prompt displayed in response to a terminal entry is I?, the Line Editor is in input mode. If the I? prompt is displayed and you want to quit the Line Editor, you must switch to edit mode. This can be done with the !F directive, as shown:

```
E? ABUILD A FILE
I? !F
```

The I? prompt indicates that the Line Editor is in input mode. The !F entry causes the Line Editor to return to edit mode, as indicated by the E? prompt. The uses of input mode and edit mode are covered in detail in this section.

Once the Line Editor is operating in edit mode, exiting from the Line Editor is accomplished with the Quit directive or Q. This directive causes the Line Editor to halt and returns you to command level.

Example:

The following example shows that the Line Editor is operating in edit mode due to the !F directive. In response to the E? prompt, enter Q to return to the command level. This return is shown by the RDY: prompt. At this point, you can execute any command or reenter the Line Editor with the ED -PT directive.

```
I?!F
E?Q
RDY:
```

### Creating A File

To create a source file using the Line Editor, invoke the Line Editor (with the -PT and -SF arguments) and invoke input mode using the I directive as shown:

```
E?I
I?
```

Once in input mode, your lines of code are entered sequentially into your current buffer. The current buffer is allocated when the Line Editor is invoked, and a pointer is established to point to this buffer as the working buffer. The current buffer is a temporary work area that is established in your working directory and memory pool (allotment). You can build a new file or read a permanent file into the current buffer for additions or modifications.

When you quit the Line Editor, the current buffer is released. Buffer management is discussed later. Each line of data entered starts in position 1 of the line and is terminated with a carriage return. When all lines of data have been entered, you terminate input mode with the !F directive. The following example shows directives used when entering data and terminating input mode:

```
E?I
I?IDENTIFICATION DIVISION.
I?PROGRAM-ID. PAYROLL3.
I?!F
E?
```

In this example, enter an I to switch to Input mode. Enter two lines of a COBOL program, pressing the carriage return after each entry. Enter !F to switch processing back to Edit mode.

As shown later, the A directive can also be used to build a new file.

### Addressing Techniques

After entering all the lines of data or source code, you may need to make corrections to a line (or lines) before saving the file. However, before you learn the directives for making corrections, it is necessary to understand basic addressing techniques. You must be in edit mode to address current buffer contents.

#### ADDRESSING A SINGLE LINE

To address (specify for access) a single line you need to enter only the line number (assigned by the Line Editor as each line was entered) followed by a directive. Or, in the case of addressing the current line, you need to enter the directive only. The current line is established either as the last line addressed in edit mode or as the last line entered in input mode.

For example, assume that you want to address line 3 to view its contents. You enter 3 followed by the P directive to view the contents of the line. The following example uses these directives.

```
E?3P
AUTHOR. NAME.
E?
```

Notice that the P follows the 3. The sequence that you enter directives is very important. The syntax rule for entering directives is:

[adr<sub>1</sub> ][,adr<sub>2</sub> ] command

Since you are working in edit mode and line 3 is the last line addressed, it is now the current line. To print line 3 again, as shown in the following example, you enter the P directive alone.

```
E?P
AUTHOR. NAME.
E?
```

#### ADDRESSING MULTIPLE LINES

To print the contents of several lines in sequence use two line numbers separated by a comma. The comma informs the Line Editor that the line numbers are inclusive. An example is 10,12P, as shown:

```
E?10,12P
DATA DIVISION.
WORKING-STORAGE SECTION.
01 PNAME PIC X(5)
E?
```

In this example the contents of lines 10, 11, and 12 are listed at the terminal.

To print the contents of the entire current buffer, use the special character \$ as the end-of-file character. When the \$ follows the comma in line number addressing, all lines from the initial value to the end of the file are affected. For example, if the addresses 7,\$ are specified with the P directive, line 7 through the end of file is listed. The following example shows a use of the \$ with the P directive.

```
E?1,$P
IDENTIFICATION DIVISION.
AUTHOR. NAME.
OTHER LINES
STOP RUN. LAST LINE
E?
```

The entry shown in this example tells the Line Editor to start with line 1 and list to the end of file (the last entry made in Input mode) the contents of all lines. The printing of the contents is caused by the P directive.

#### PRINTING LINE NUMBERS

Note that when the P directive is used, only the contents of the line (or lines) specified is shown. To print specified contents of the current buffer and display the line numbers assigned to each line, specify !P.

The following example shows the use of the !P directive. Notice that the only difference in the listing is that line numbers are displayed at the terminal with the buffer contents.

```
E?1,$!P
1 IDENTIFICATION DIVISION.
2 PROGRAM-ID.  PROGRAM.
3 AUTHOR.  NAME.
```

E?

#### USE OF PERIOD (.) FOR CURRENT LINE

If you use multiple line addressing and want to start with the current line, use the period (.) character in the first address.

For example, your current line is 10 and you want to print through line 20. If the !P directive were used instead of the P directive in the following example, lines 10 through 20 would be listed with the file contents.

```
E?.,20P
OBJECT-COMPUTER.  HIS-SERIES-60 LEVEL 6.
DATA DIVISION.
OTHER LINES
MOVE "DATUM" TO QNAME.
E?
```

#### CHARACTER STRING ADDRESSING

The Line Editor can also address a line (or lines) by contents.

The contents, called a character string, are expressed using two delimiters. Delimiters are slashes (/) that precede and follow the designated search string. For example, the slashes in the expression /ABC/ delimit the string ABC. The two slashes are the delimiters, and the characters between are the string that is searched for. The search begins with the line following the current line, continues to the end of the file, and then starts with line 1 and searches to the current line. When the Line Editor finds a line containing the specified string it executes any directive specified with the search string and positions the current line pointer to that line.

In the following example, enter PROGRAM-ID as a character string with delimiters, with the P directive. As a result, the line containing PROGRAM-ID is printed, revealing that the internal name of the program is PROG1. If the !P directive were used, the line number would have been printed with the line contents.

```
E?/PROGRAM-ID/P
PROGRAM-ID.  PROG1
E?
```

The search for the contents ends when the first line that contains the specified character anywhere in that line is found. The current line pointer is then positioned at that line. If no match is made with the search string, the current line pointer is positioned at the line that was the current line before the directive was executed. Also, if there is no character string match, the message SEARCH FAILED is displayed at the terminal.

### Selective Specification of Character Strings

To be more selective when specifying search characters, make the string within the delimiters more specific. For example, you may want to print the line number and the contents of a line that contains the character X. Suppose there are two lines that contain X. One contains PIC X(5) and another PIC X(10). To specify the first, you designate the search string so that the Line Editor can determine that you want the line that contains PIC X(5). You could specify PIC X(5) as the character string. Or, you could specify only that portion of the string that distinguishes it from all other strings in the file. The string X(5 would be sufficient. If you enter /X(5/!P at the terminal, the only line that qualifies for printing is PIC X(5).

In the example, only DATA DIVISION is printed because others, like PROCEDURE DIVISION, do not fit the search characteristics.

```
E?/A DIVISION/!P
11 DATA DIVISION.
E?
```

### Specifying Initial Character String

The Line Editor can also be told to search for a line beginning with a character string. This is done by preceding the character string with a circumflex (^). For example, assume that the specified character string TAG must occur as the first characters on a line. The following example shows specification of this search string:

```
E?/^TAG/!P
2 TAG EQU 10
E?
```

In this example the use of the circumflex (^) specifies the character string TAG must occur as the first three characters of the line searched for. The !P directive is used to print line contents with the line number.

### Specifying a Character String Ending a Line

A string occurring as the last character(s) on a line can be specified as a search string. You specify this with the \$ character. To specify that you want to find the line that ends with the characters FILE, use the search string: /FILE\$/.

The following example shows the use of the \$ character to specify the FILE ending character string with the !P directive. The result is the terminal listing of FD INFILE with the line number 32.

```
E?/FILES!/P
32 FD INFILE
E?
```

Remember that the results of the directive shown in the previous example are dependent upon the location of the current line. If the current line was after line 32 in the buffer (at the time of execution of the directive in the previous example), the terminal display might contain a different line number and contents, such as FD OUTFILE. Therefore, it is important to know the location of the current line when you initiate a character string search.

As shown in the addressing methods, certain characters represent special or control characters. The circumflex (^) is used to specify a line beginning with a specified string. The \$ is used to specify a string ending a line when it is used in a search for line contents. The dollar sign (\$) is also used to specify the end of file when it is entered as the second address to designate multiple lines as in this example. The period (.), when used in line number specification, represents the current line. It has another use--single character substitutions.

#### Specifying a Single Character Substitution In Search Strings

When a period (.) is used in a character string search, it takes on a special meaning. When a period is used in a search string such as /A.C/ it means that any character can be substituted between the characters A and C. This means that ABC fits the search, as does AIC or AZC.

In the following example, EDB is the first search match, so line 10 is printed.

```
E?/E.B!/P
10 LABEL EDB $B5, X'FFFF'
E?
```

#### Use Of Escape Characters

It is possible that any of the special addressing characters (\$, !, or ^), are part of the data to be searched for. The preceding example contains such an example in the display of line 10.

To distinguish between a special character as data and a special character used to affect a search string, escape characters are used. For example, to specify that the character has its data meaning and not search meaning, !C escape characters are used.

The escape characters remove the search meaning from the next character. For example, the search string /Al!C\$/ contains !C, which removes the search meaning from the \$. The Line Editor searches for three characters designated as Al\$ rather than Al at the end of a line. The following example shows another use of the escape characters.

```
E?/ION!C.$/!P
1 IDENTIFICATION DIVISION.
E?
```

In this example, !C precedes the period so that IDENTIFICATION DIVISION. is found. Also, the \$ symbol ensures that ION. occurs in the last four characters of the line that is found and listed.

### Saving File Contents

All of the work done building a file in the current buffer is destroyed when you quit the Line Editor. The current buffer is a temporary working file. The contents of the current buffer must be stored in a permanent file if the buffer contents are to be saved after you quit the Line Editor.

If you are building a new file in the current buffer, you need to create a new permanent file to accept the contents of the current buffer. A new file can be created using the CR command at the command level before you call in the Line Editor. The current buffer contents can be copied into the previously created file. If the file you want to create is to be a sequential file, you can create that file at the same time that you copy current buffer contents by using the W directive.

As shown in the following example, a COBOL source program has been built in the current buffer. The file (source program) is saved to a permanent file called COBOLP.C. The file named COBOLP.C did not exist before the W directive was entered. When the W directive is entered, the file named COBOLP.C is created as a sequential file immediately subordinate to the working directory (however, a full or relative pathname could have been used). After the W directive creates the file, the same directive copies the contents of the current buffer to the designated permanent file, which is COBOLP.C in this case.

```
E?W COBOLP.C
E?
```

Two other situations exist in which you may want to save the current buffer contents to a permanent file. One situation was already mentioned. The file may exist before the W directive is used because the file was created using the CR command. Another situation is when you want to replace the contents of a file with the contents of the current buffer, as in the case of making modifications to a program. In either case, the W directive stores the contents of the current buffer in the designated file.



In the case of the file that was created using the CR command, the contents of the current buffer are copied into the existing permanent file. In the case of the permanent file that exists and contains a previously stored program or data, the old file contents are replaced by the contents of the current buffer. The format for copying the current file contents to the existing permanent file is the same as the one shown in the preceding example. Just be sure to designate the correct pathname for the existing permanent file that is to contain the current buffer file.

After preserving the contents of the new file you can quit the Line Editor, return to command level, and perhaps compile the program, if this is a source program. Remember, if you did not write the buffer file to a permanent file prior to quitting the Line Editor, all data from the editing session is lost.

### Reading File Contents

This subsection describes the procedures used to modify the contents of a new file, or to modify a source or data file already in existence.

Invoke the Line Editor and be sure you are working in Edit mode.

If you want to modify the contents of an existing file, you must copy that file into the current buffer before you can use the file modification techniques.

If the file to be modified exists as a permanent file, you must copy the file contents into the Line Editor's current buffer. This is done with the Read (R) directive followed by the pathname of the file to be altered.

Note that in the following example, a full pathname is specified. You can use any of the pathname variations allowed.

```
E?R^VOLA>DIR1>COBOL.C
E?
```

The R directive is a read and append directive. That is, the contents of the file read are appended to the contents of the current buffer. If you want the contents of the file being read as the only data in the current buffer, first delete the contents of the current buffer and then perform the read. If the Line Editor was just invoked, the sequence is:

```
ED -PT
E? R pathname
```

## Deleting Lines In Current Buffer

The D directive is used to delete lines from the current buffer. To delete lines, specify the line (or lines) to be deleted followed by the Delete directive (D). To delete one line, use the line number followed by D, as shown:

```
E?5D
E?
```

To delete the contents of the current line specify just the directive D, as shown:

```
E?D
E?
```

### DELETING MULTIPLE LINES

To delete multiple lines in sequence, specify the line numbers separated by a comma and followed by the D directive.

In the following example, 5,10D causes lines 5 through 10 to be deleted from the current buffer:

```
E?5,10D
E?
```

### DELETING ALL LINES IN CURRENT BUFFER

To delete all lines in the current buffer, use the character \$ as the second address and line 1 as the first address.

The following example shows the directive sequence (1,\$D). This directive sequence is used to delete or clear the contents of the current buffer for the use of the R directive explained under "Reading File Contents." There are times when you do not want to clear the current buffer before using the R directive; these instances are covered later in this section. Usually, however, you do want a clean current buffer before you read a permanent file into it.

```
E?1,$D
E?
```

The following example shows a typical clear-and-read sequence. The 1,\$D directive clears the current buffer. Then the file named COBTEST, immediately subordinate to the working directory, is read into the current buffer.

```
E?1,$D
E?R COBTEST
E?
```

## AVOIDING POST-DELETION PROBLEMS

In the preceding examples, the use of line numbers to specify lines to be deleted was shown. After the line is deleted any remaining lines following the deleted line are automatically renumbered. This can cause problems in the deletion or modification of remaining lines.

For example, if you delete line 10, what was line 11 is now line 10 and line 12 is now line 11 and so on through the end of the file. If your next directive affects old line 15, you must remember that it is now line 14.

Example 2 shows the results of deleting lines 2 and 3 from the current buffer file shown in Example 1. Note that line 4 (containing the PROGRAM-ID) becomes line 2. All subsequent line numbers are affected in the same way. Notice that line numbers before the deleted line(s) are not affected; the line number for the IDENTIFICATION DIVISION statement in this case does not change.

Example 1:

```
E?1,$!P
1 IDENTIFICATION DIVISION.
2 ****NOTE: MAKE SURE THAT YOU CONFIRM INSTALLATION,****
3 ****AND SECURITY BEFORE COMPLETING PROGRAM*****
4 PROGRAM-ID. COURSE.
5 AUTHOR. AMY SMITH.
6 INSTALLATION. LOMPOC, CA.
7 DATE WRITTEN, 05181
8 SECURITY. NONE.
9 ENVIRONMENT DIVISION.
10 CONFIGURATION SECTION.
11 SOURCE-COMPUTER. HIS-SERIES-60 LEVEL-6.
12 OBJECT-COMPUTER. HIS-SERIES-60 LEVEL-6.
E?
```

Example 2:

```
E?2,3D
E?1,$!P
1 IDENTIFICATION DIVISION.
2 PROGRAM-ID. COURSE.
3 AUTHOR. AMY SMITH.
4 INSTALLATION. LOMPOC, CA.
5 DATE-WRITTEN. 05181
6 SECURITY. NONE.
7 ENVIRONMENT DIVISION.
8 CONFIGURATION SECTION.
9 SOURCE-COMPUTER. HIS-SERIES-60 LEVEL-6.
10 OBJECT-COMPUTER. HIS-SERIES-60 LEVEL-6.
E?
```

## ADDING AND DELETING LINES

To avoid confusion when adding or deleting lines, use one of the following methods. The first is to delete or add lines starting with the line nearest the end of file. For example, you must delete lines 10, 15, and 20 in a buffer containing 25 lines. If you start with line 20 first, lines 10 and 15 are unaffected because only the lines following 20 are renumbered. Then line 15 should be deleted so that line 10 is not affected by the change. Finally, line 10 should be deleted.

The second method is to delete by contents of a line. For example, you know that only the line containing STOP RUN is to be deleted. So, you specify a search for the line by its contents with the D directive.

The following example shows how a line can be deleted, regardless of its line number. Naturally, any number of lines can be deleted using this method, but a separate D directive must be used for each line to be deleted.

```
E?/STOP RUN/D
E?
```

Due to automatic resequencing of line numbers after line additions or deletions, it is helpful to list the file to determine the new line numbers before you attempt any subsequent modifications by line number.

### Changing Line Contents

To change the contents of a line, use the Change (C) directive or the Substitute (S) directive. The C directive allows you to change the entire contents of a line. This directive is considered an input mode directive. After executing the C directive the Line Editor is in input mode. This requires that you execute the !F directive to exit input mode.

In the following example, the entire new line had to be entered. You cannot use the C directive to change just portions of a line. Also, the use of the !F directive at the end of the entry causes the Line Editor to return to edit mode.

```
E?10!P
10 MOM'S APPLE PIE.
E?10CSOURCE-COMPUTER. LEVEL-6. !F
E?
```

### CHANGING CHARACTER STRINGS WITHIN A LINE

To change portions of a line, use the Substitute (S) directive. The S directive allows you to substitute one character string for another within a line. The S directive is an edit mode directive.

To make a substitution, enter a line number, an S, a delimiter, the character string to search for, a delimiter, the substitute string, and a final delimiter.

To substitute a new character string for a specified character string on line 5, you could use this method:

```
E?5!P
5 PROGRAMMER-ID. ROGER.
E?5S/ROGER/GARY/
E?
```

In this method all occurrences in the line of the character string ROGER would be replaced by the character string GARY. This means that if ROGER occurs three times on line 5, three substitutions are performed.

To specify that only one occurrence of a character string is to be altered, you must be more specific in the search character string. For example, the word ROGER occurs only once in the line shown in the preceding example. If ROGER occurs twice in the affected line and you want to change only the first occurrence to GARY, the search character string must be made more specific. This is described in the earlier section, "Selective Specification of Character Strings."

#### CHANGING ALL OCCURRENCES OF A STRING

The Line Editor can be told to search one line, multiple lines, or all lines for a specified character string and to substitute all occurrences of that string for the new string.

In the following example, all occurrences of IONFILE in the current buffer are changed to INFILE:

```
E?1,$S/IONFILE/INFILE/
E?
```

Techniques for substituting, printing, and deleting all occurrences of a specified string are discussed below. These techniques for affecting all occurrences of a string can be helpful in program modification.

For example, a program that already exists might be almost perfect to meet a new data processing need, but the record or field names are wrong. With a single directive, all occurrences of the unwanted record name or identifier can be changed to the necessary entry.

#### SUBSTITUTING INITIAL AND CONCLUDING STRINGS

As explained previously, the circumflex (^) and dollar sign (\$) can be used in an address search. Those same rules apply here. To specify a character string that begins a line, use the ^ character.

The following example shows a directive used to substitute the character string PROGRAM for the occurrence of PRGRAM at the beginning of line 3. PROGRAM replaces PRGRAM as the initial character string. To affect a string that ends the line, use the \$ character.

```
E?3S/^PRGRAM/PROGRAM/  
E?
```

The next example shows the concluding character string OUTFILD at the end of line 7 is replaced by the string OUTFILE., through the S directive. Of course, it is possible to replace all occurrences of an initial or concluding string with a specified character string.

```
E?7S/OUTFILD$/OUTFILE./  
E?
```

In the following example, starting with line 1 through the end of the file, all occurrences of the # at the beginning of a line are replaced by seven spaces.

```
E?1,$S/^#/ /  
E?
```

The technique shown in this example is commonly used in editing source programs that by convention have coding statements beginning in specific columns of a line. Column 7 in FORTRAN and columns 8 and 12 in COBOL are examples of columns that are specified for coding purposes. For example, all statements that should start in column 8 might begin with a # character. As a result of the use of the directive in the example, all of those statements now begin in column 8 because of the preceding seven spaces.

#### DELETING CHARACTER STRINGS

If you have entered a character string and later find that you do not need it, you can delete that string using the substitute directive. To delete a character string specify the substitution field as blank.

In the following example, the substitution field is specified as //. This informs the Line Editor that there is no replacement string. Therefore, the string specified in the search field (DATA) in line 20 is to be deleted.

```
E?20S/DATA//  
E?
```

## APPENDING A NEW STRING TO AN EXISTING STRING

Expanding a character string on a line (or lines) can be done with the ampersand (&). The position of the & dictates where the new character string occurs in the new line. For example, the character string L6 is on line 23, and that character string should be LEVEL-6. The following example shows how the use of &, by its position, changes the line in the correct location.

```
E?23!P
23 OBJECT-COMPUTER. HIS-SERIES-60 L6.
E?23S/L/&EVEL-/
E?23!P
23 OBJECT-COMPUTER. HIS-SERIES-60 LEVEL-6.
E?
```

The substitution causes the L in L6 to be followed immediately by EVEL-. The result is LEVEL-6. Notice that there are no spaces between delimiters and strings or between special characters (such as & or ^) and the strings. It is important to define the search strings and the modification strings accurately, or the result will be incorrect.

The escape characters (!C) cause an editing character to have no meaning to the Line Editor. For example, if the character & is to be used as a nonediting character in the second field of a Line Editor directive, the & should be preceded by the !C escape characters. Such an example would occur if you want to change the string \$ION to &ION. The line entry would be S/!C\$ION/!C&ION/.

## ADDING LINES TO THE CURRENT BUFFER

To add a new line to your current buffer, use either the Append (A) directive or the Insert (I) directive.

### Inserting Lines

Using the I directive you can insert a line (or lines) before the line specified in the address. To insert a line of code preceding line 15, enter 15I.

The results of the directives shown in the following example are a new line 15 and all subsequent line numbers incremented by 1. The old line 15 becomes line 16, and so forth.

```
E?15I WORKING-STORAGE SECTION. !F
E?
```

Notice in this example that the E? prompt is displayed after you press the carriage return. In this case, because the !F directive follows the entry of the new line, the E? prompt signals that the Line Editor is ready to work in edit mode. If the !F directive were not entered, the I? prompt would be a request for another line of input. Remember, the I directive causes the Line Editor to work in input mode.

Two lines are inserted in the file in the current buffer in the next example. The first line added becomes line 15 because of the 15I directive. The I? prompt requests another line of input. The RBN-STATUS line is entered. It becomes line 16. The I? prompt requests another line of input. The !F response causes a switch to edit mode.

```
E?15I WORKING-STORAGE SECTION.  
I? RBN-STATUS PIC X.  
I?!F  
E?
```

Remember, because lines 15 and 16 are inserted as new lines, the old line 15 becomes line 17 and all subsequent line numbers are incremented by 2.

### Appending Lines

To append a new line to any point in a file, the A (Append) directive is used. Append adds a new line following the number specified.

The example below shows the use of the Append directive. This directive creates a new line 16 and all subsequent lines are renumbered. The new line follows line 15. The A directive allows you to add more than one line. If the !F directive was not used in this example, the Line Editor would be expecting you to enter another new line that would be line 17.

```
E?15A #WORKING-STORAGE SECTION. !F  
E?
```

Note that after the A directive, the !F directive is needed to exit input mode.

### Global Directives

Searching for lines that need to be modified or listed can be simplified by using the Global directive. The Global directive (G) works only with the following directives: P, !P, D, and =.

With the Global directive only the lines that contain the specified character string have the directive P, !P, D, or = applied to them.

### GLOBAL DELETE

Global Delete (GD) is used to remove a character string throughout a file. To issue a Global Delete, type the directive G followed by D, followed by the character string to search for in delimiters. The following example shows a Global Delete.

```
E?GD/DATA/  
E?
```



In this example, no line numbers are specified. Line numbers can be specified, but when they are absent, the directive defaults to start at line 1 and works to the end of the file. If line numbers are specified, only those lines specified are affected by the delete. For example, the entry 1,15GD/DATA/ removes all lines containing the string DATA from lines 1 through 15.

## GLOBAL PRINT

To print only lines that contain certain characters, specify the GP or G!P directive.

The GP directive prints just the contents of the lines that contain the character string and the G!P directive prints the contents and the line numbers associated with those contents.

The following example shows the terminal entry requesting the line number for each of the four COBOL divisions for a program that is in the current buffer. The G directive with the !P directive and the SION search directive is entered. The SION has been used as the search string because it is common to all four COBOL program divisions.

```
E?G!P/SION/  
 3 IDENTIFICATION DIVISION.  
 7 ENVIRONMENT DIVISION.  
10 DATA DIVISION.  
21 PROCEDURE DIVISION.  
E?
```

The next example shows the G and = directives with the same search and current buffer file used in the previous example. Note that only line numbers are printed.

```
E?G=/SION/  
 3  
 7  
10  
21  
E?
```

## Current and Auxiliary Buffers

Thus far, changes have been made to a permanent file through the use of the current buffer. In addition to the current buffer, there are five auxiliary buffers available to assist in manipulation of file contents.

For example, to repeat lines of coding in a program, to move lines of coding from one location in a file to another location, or to build a new file from coding lines of other files, auxiliary buffers are used.

## REPEATING LINES IN A FILE

There are a number of file-building and file-modification functions that can be carried out through buffer management. The current buffer and up to five auxiliary buffers assist in creating or modifying file contents. The auxiliary buffers can have alphabetic or numeric names. The examples shown below use single number names.

The next two examples illustrate buffer manipulation used to repeat lines in a file. The directives used in the manipulation of current and auxiliary buffers are the K and !B directives. K and !B are used for copying lines to a specified auxiliary buffer and for fetching lines from a specified auxiliary buffer.

The following example shows how the current buffer and an auxiliary buffer can be used to repeat lines of a file at the end of the file. The program stored in the file named COBPRG.C is read into the current buffer, and lines 50 through 63 are copied to an auxiliary buffer named 1, through the K1 directive. If any lines exist in the auxiliary buffer at the time that the K directive is used to copy new lines to that buffer, the old lines in the buffer are deleted before the new lines are copied to that buffer.

```
RDY:
ED -PT
E?R COBPRG.C
E?50,63K1
E?$A!B1!F
E?W COBPRG.C
E?1,$D
E?
```

After the copy is completed, the lines in buffer 1 are appended to the end of the file in the current buffer. The \$A directive causes lines to be appended to the end of the file. The !B1 directive causes the lines in buffer 1 to be fetched for appending. The !F directive is used to change to edit mode (after the A directive is used). Then the W directive causes the current buffer contents to be copied to the file named COBPRG.C to replace the old contents in that file. Finally, the contents of the current buffer are deleted with the D directive to allow a new file to be read into the current buffer. If no more editing is to be done, the current buffer contents do not have to be deleted. The Q directive would be sufficient to quit the Line Editor.

The next example demonstrates two editing concepts important to buffer manipulation--how to repeat lines within a file and how to use a different auxiliary buffer to save the contents in an existing auxiliary buffer.

```
E?R CFILE.C
E?9,17K2
E?45A!B2!F
E?W CFILE.C
```

A file named CFILE.C is read into the current buffer, and lines 9 through 17 are copied to an auxiliary buffer named 2. If there are lines in another auxiliary buffer, such as buffer 1, the copying of lines to buffer 2 allows the lines to remain in buffer 1. The lines in buffer 2 are then appended to line 45 in the current buffer (through the 45A directive). The !B2 directive causes those lines to be fetched from buffer 2. Again the !F directive causes a return to edit mode. Finally, the changes must be made to the file named CFILE.C through the W directive.

#### MOVING LINES IN A FILE

The next example shows moving lines within a file. The first situation involves moving lines to a location near the end of the file.

The following example shows how lines 8 through 12 in a file can be moved to follow line 27. A file named FILEN is read into the current buffer. Lines 8 through 12 are copied to buffer 1. Then the lines are appended to line 27 in the file. Because the lines in the original location remain, they must be removed through the D directive. Then the current file contents are written back to FILEN. The current buffer contents are deleted to allow for additional editing of a different file.

```
RDY:
ED -PT
E?R FILEN
E?8,12K1
E?27A!B1!F
E?8,12D
E?W FILEN
E?1,$D
E?
```

When lines are moved to a location closer to the beginning of the file than their current location, the deletion of the old lines after the move presents a different problem.

The next example shows the steps taken to move lines 21 through 37 of a file to the beginning of the file. The file named RNGT is read into the current buffer, lines 21 through 37 are copied to buffer 1, and lines 21 through 37 in the current buffer are deleted.

The I directive is used to insert the contents of buffer 1 before line 1 of the file. The file is listed. The current buffer is written to RNGT.

The Line Editor keeps track of the actual line numbers of the lines in a buffer. You can add, delete, or rearrange lines and the Line Editor automatically rennumbers to keep the count correct.

```
E?R RNGT
E?21,37K1
E?21,37D
E?11!B1!F
E?1,$!P
```

file is listed

```
E?W RNGT
E?
```

#### USING EXISTING FILES

Sometimes a new program must be created and it contains logic elements that are similar to elements in one or more existing programs. In this case, you can call an existing program into the current buffer, delete the unnecessary coding from the current buffer, and build a new file around the useful coding. At other times, you may want to call portions of different programs into the current buffer and use those portions as a basis for building a new program. These Editor and buffer techniques can save program development time.

The following example shows how two programs can be used for developing a single new program. In this example only two buffers are used. It should be noted that all five can be used. Additionally, if necessary, the entire contents of the current buffer can be copied to an auxiliary buffer as different portions of the program are developed. This example shows the steps that can be used to develop the program from two existing programs.

|               |                                       |
|---------------|---------------------------------------|
| E?R ABPRG.C   | Read in ABPRG.C.                      |
| E?1,8K1       | Copy lines 1 through 8 to buffer 1.   |
| E?59,67K2     | Copy lines 59 through 67 to buffer 2. |
| E?1,\$D       | Delete contents of current buffer.    |
| E?R GGPRG22.C | Read in GGPRG22.C.                    |
| E?29,32D      | Delete lines 29 through 32.           |
| E?26,27D      | Delete lines 26 and 27.               |
| E?1,12D       | Delete lines 1 through 12.            |
| E?1,\$!P      | List file with line numbers.          |

listing is produced

|           |  |
|-----------|--|
| E?5A!B2!F | Append contents of buffer 2 after line 5.  |
| E?11!B1!F | Insert contents of buffer 1 before line 1. |
| E?1,\$!P  | List file with line numbers.               |

listing is produced

```
E?W NEWP.C    Write to a permanent file.
E?
```

The lines are deleted from the end of the file (29,32D) toward the beginning of the file (1,12D). The next step shown in this example is the listing of the current buffer at the terminal. This listing indicates line numbers for the insertion of lines from the two auxiliary buffers. The lines in buffer 2 are appended to line 5 of the current buffer (5A!B2!F). If the lines from buffer 1 were inserted at the beginning of the file, the line numbers in the current buffer would change. Then you would have to list the contents of the current buffer again to see where the lines from buffer 2 should be appended. After the lines from buffer 2 are appended to line 5, the lines from buffer 1 can be inserted before line 1 of the current buffer.

To make sure that the current buffer contains all the lines in the proper sequence, list the file contents again. Then you can write the current buffer contents to a permanent file or continue working on the current buffer contents to create a program. The contents of the current buffer are saved to a permanent file named NEWP.C.

#### BUFFER STATUS

The X directive is used to determine buffer size or current buffer status. Buffer status can be checked at any time during the editing process. The following example shows the use of the X directive:

```
E?X
15->(0)^VOL10>EDMOD>COBPRG.C
E?
```

The example indicates that there are 15 lines in buffer 0. The second field of the display information is an arrow pointing to the buffer that is the current buffer. In this example, only one buffer is shown; the following example shows two buffers.

```
E?X
15->(0)^VOL10>EDMOD>COBPRG.C
8   (1)
E?
```

This example indicates that there are two buffers; one with 15 lines and one with 8 lines. The buffer pointer is pointing at buffer 0, the current buffer.

If you were to add, delete, change, or substitute lines in the current buffer, MOD would appear, as shown:

```
E?X
15  MOD->(0)^VOL10>EDMOD>COBPRG.C
8   (1)
E?
```

The next field is the buffer name in parentheses. The name for the first is 0 (the default current buffer), which was created when the Editor was invoked. The second is 1, which was created when data was moved into it.

The final field in the buffer status is the absolute pathname used in the last read or write operation involving that buffer. Notice in the preceding examples that the file with the absolute pathname ^VOL10>EDMOD>COBPRG.C has been read into the current buffer.

#### SAVING MODIFIED BUFFER CONTENTS

The following example shows part of a terminal session where a file has been modified using the Line Editor. You attempt to quit the Line Editor with the Q directive. Because you have not saved the current buffer contents with the W directive, the system displays the QUIT DEFERRED message. This message indicates that modifications have been made to the current buffer but the current buffer contents have not been saved.

```
E?Q
MODIFIED BUFFERS EXIST, QUIT DEFERRED
E?W TESTFL
E?Q
RDY:
```

In this case, you respond to the message with the entry W TESTFL and to the E? prompt with the Q directive to return to the command level of processing. If you responded to the QUIT DEFERRED message with the Q directive, the system would have accepted the directive, the current buffer contents would have been destroyed, and processing returned to command level.

#### Using System Commands in the Editor

The Execute directive (E) allows you to use system commands while you are working with the Line Editor.

The Line Editor must be in edit mode. Then an E followed by any system command causes the Line Editor to pass that command to the Command Processor. After executing that command, the system returns control to the Line Editor.

#### WRITING TO LINE PRINTER

To write the contents of the current buffer to the line printer, first reserve the line printer as the user output file. This is done with the FO command explained earlier. The following example shows the commands used to reserve a line printer.

```
E?E FO !LPT00
E?
```

After executing the command shown in the example, the Line Editor sends all output that would go to the screen (except for the ready display and errors) to the line printer. To get hard copy of the current buffer, type the directive line that prints the entire contents of the buffer.

The next example shows that once user output is directed to a printer, any variation of the P directive causes printing to take place at the printer. In this case, the entire file (1,\$) is printed with line numbers (!P).

```
E?E FO !LPT00
E?1,$!P
E?
```

After printing the contents of the current buffer on the line printer, to change output back to the terminal (or default device), enter the FO command with no pathname.

The following example shows the Execute (E) directive is necessary to execute the FO command while the Line Editor is invoked.

```
E?1,$!P
E?E FO
E?
```

#### DATE AND TIME

When the lines of the current buffer are listed on the line printer they are printed as is. There is no date or time displayed with the printing. Sometimes it is helpful or important to know when a listing of a file was made, particularly when various updates of files must be compared.

To display a date and time heading with your listing on the line printer, use the system command TIME after directing output to the line printer.

TIME is a system command. Therefore, the Execute (E) directive must be entered too.

The directive line E TIME causes the system date and time to be displayed on the output device. This display becomes a header for the file listing that follows. For example, as a result of the following entries, the entire contents of the current buffer are listed at printer LPT00 with a date and time header.

```
E?E FO !LPT00
E?E TIME
E?1,$!P
E?
```

## IMPORTANT CONSIDERATIONS

When using the Execute (E) directive you can execute any system command. However, if the E directive is omitted, certain problems can occur. For example, if the E directive is not used, the Line Editor does not pass the entry to the Command Processor. As a result, the Line Editor tries to execute the entry as a Line Editor directive. For this reason, accidentally entering a command to the Line Editor without the E directive can cause problems. Accidentally entering a command that begins with any of the following characters can be particularly problematical: W, LW, D, I, C, and A.

For example, an entry beginning with a C changes line contents. An I or an A causes additions to a current buffer file. A W copies the current buffer contents to a permanent file. Therefore, using the E directive is important.

For instance, if LWD is entered without the preceding E, the Line Editor causes a line feed and writes the contents of the current buffer to a file called D under the working directory. The correct way to enter the LWD command is shown in the example:

```
E?E LWD  
^VOL3>DIR23  
E?
```



REMOVE THIS PAGE AND PLACE TAB FOR :

TAB 6

LINKER



## Section 6

# ***LINKER***

This section describes Linker functions and the Linker directive set. Procedural information on using Linker directives to create a bound unit is also included in this section.

### LINKER FUNCTIONS

The Linker combines object units created by the language processors (compilers and the Assembler) into a bound unit that you can then execute. During a single execution of the Linker, a single bound unit is created. A bound unit contains a root or a root with one or more overlays.

The Linker functions are:

- CREATE A BOUND UNIT--A bound unit is the output file that results from Linker execution. The bound unit is an executable program.
- BUILD A SYMBOL TABLE--During the linking process, the Linker builds an internal symbol table used for resolving external references (i.e., references within an object unit to locations or values outside that object unit). You can define a symbol within an object unit or by using Linker directives described later in this section.

- PRODUCE A LISTING--The Linker listing has two parts; a dynamic part and a static part.
  - The dynamic part is generated continuously and contains information about each object unit linked, the directives used, and a summary.
  - The static part is produced in response to one of the MAP directives ( MAP, MAPD, or MAPU) and is a picture of the state of the link when the directive is processed. It lists the external definitions currently in the symbol table and the unresolved references to external symbols, if any exist.

At the end of the link process, summary information about the bound unit is automatically output to a list file. The format of this information is as follows.

| LINK SUMMARY  |        |            |       |       |       |        |
|---|--------|------------|-------|-------|-------|--------|
| All values are in hex   |        |            |       |       |       |        |
| Load Unit Description:  |        |            |       |       |       |        |
| Name  | Number | Attributes | Base  | Start | Size  | Access |
| *DATA   |        | DF         | E0000 | 00000 | 000C8 | 000    |
| JAN10   |        | R          | 20000 | 20028 | 0003A | 000    |
| **FIRST   | 0      | O          | 2003A | 20053 | 0002B | 000    |
| Key to Attributes:  |        |            |       |       |       |        |
| R=Root; D=Data; O=Fixed overlay; F=Floating overlay               |        |            |       |       |       |        |
| U=Contains unresolved references; I=Contains IMAs                 |        |            |       |       |       |        |
| Bound Unit Description:   |        |            |       |       |       |        |
| Linked for BMMU   |        |            |       |       |       |        |
| Size of fixed area: 65  |        |            |       |       |       |        |
| Number of overlays; Fixed: 1, Floating: 1, Total: 2               |        |            |       |       |       |        |
| Uninitialized data area; Size: C8, Initialization value: 00000000 |        |            |       |       |       |        |
| Bound unit record; size: 100, count: 2                            |        |            |       |       |       |        |
| LINK DONE   |        |            |       |       |       |        |

- RESOLVE EXTERNAL REFERENCES--The Linker resolves external references in object units being linked. To do this, the Linker uses external definitions found in object units or declared by LDEF, VDEF, or VAL directives. (These directives are described later in this section.) When a bound unit is linked, any unresolved external references are listed at the end of the link map. If unresolved external references exist at the end of the Linker run, an error message is displayed on the error-out file (usually the terminal).

---

\*This line appears only if common has been gathered into one contiguous area. The -R control argument was specified in the Linker command line.

\*\*This line repeated for each overlay.

## LINKER DIRECTIVE CATEGORIES

The Linker directive set can be grouped into the functional categories described in the following paragraphs.

### Specifying Object Unit(s) to be Linked

LINK, LINKN, LINKnn, and LINKO designate that one or more specified object units are to be linked. Object units specified in LINK directives are not linked immediately; their names are put into a link request list. Linking begins when the Linker finds a directive that requires all preceding link requests to be honored. Specified object units in the primary input directory are linked before specified object units in the secondary input directory; within each directory, the object units are linked in the order in which they were requested.

LINKN causes the Linker to link object units already named in the link request list, and then to link object units specified in the LINKN directive in the order in which they were requested.

LINKO performs in the same manner as LINKN, except that all embedded directives in the named object unit(s) are ignored by the Linker. LINKnn is a special form of LINKN used to perform selective linking.

### Specifying Location(s) of Object Unit(s) to be Linked

Object units to be linked must be in at least one directory. The Linker searches the primary directory first, then searches other directories if they have been specified by directives described below. When the Linker is loaded into memory, the primary directory is the working directory. The directives used to specify location(s) of object unit(s) to be linked are listed below.

- IN is used to designate a directory other than the working directory as the primary directory.
- LIB is used to designate a directory as the second directory to be searched.
- LIB2 is used to designate the third directory to be searched.
- LIB3 is used to designate the fourth directory to be searched.
- LIB4 is used to designate the fifth directory to be searched.
- LSR is used to request a list of the directories in the order in which they are to be searched.

## Creating a Root and Optional Overlay(s)

START is used to specify the relative address at which the root or overlay begins executing when it is loaded into memory.

BASE is used to define a relative address (within the bound unit) where subsequent object units are to be linked. Note that once the lowest address of a root or overlay has been established (i.e., an object unit has been linked), it is invalid to define a lower BASE address within the root or overlay.

OVLY is used to name the nonfloatable overlay that follows and designates the end of the preceding root or overlay.

FLOVLY is used to name the floatable overlay that follows and designates the end of the preceding root or overlay.

FSEG is used to specify a base segment number and the access rights for a floatable overlay.

CC permits a COBOLA program that uses CALL and CANCEL statements to call overlays by their names.

IST is used to identify the beginning of initialization code in the root.

INIT2 is used to specify a 2-word initialization pattern for all otherwise uninitialized common blocks. This directive is meaningful only if a bound unit is being linked with the -R option (separate code and data).

SHARE is used to designate that the bound unit is shareable within a memory pool.

QUIT is used to designate that the last Linker directive has been entered. Execution of the Linker terminates after the bound unit has been created.

FLOATB6 is used to suppress certain error checking on local common references when the -R Linker argument has not been specified. Local common references are relocated as if B6 pointed to the base of the containing overlay, or to base plus 32K words if the overlay is that large.

STACK is used to specify the size of the stack area.

GSHARE is used to specify that the bound unit is globally shareable. Its root is loaded into the system memory pool.

SEG is used to specify nondefault segment number(s) and access rights before linking begins.

REPORT is used to ascertain the current segment number and access rights before linking begins.

SYS is used to designate that the bound unit may be run as a system task.

USER is used to designate that the bound unit must be loaded into the user area of memory.

SWAPPOOL is used to designate that the bound unit must run in a swap pool or page pool environment.

PAGEPOOL is used to designate that the bound unit must run in a page pool environment.

PSU is used to indicate that each floatable overlay will be loaded into group work space (GWS) rather than being assigned to the segment address established for it by the Linker.

ONECPU is used to designate that the bound unit and all its subordinate tasks must run in a single central processor environment.

NOTCMD is used to designate that the bound unit cannot be executed as a command.

LINK, LINKN, and LINKO are used to specify those object units to be linked. The order in which specified object units are linked, and when they are linked, is determined by the link directive used.

LDEF is used to assign a relative location to an external symbol. When a symbol is defined, its definition is put into the Linker symbol table so that it can be used to resolve references to the symbol during linking.

VDEF is used to assign a value to an external symbol. When a symbol is defined, its definition is put into the Linker symbol table so that it can be used during linking to resolve external references.

### Producing Link Map(s)

MAP is used to create a map that lists both externally defined symbols and unresolved references to external symbols.

MAPU is used to create a map that lists only unresolved references to external symbols.

MAPD is used to create a map after common block sizes have been changed to addresses. This directive is meaningful only if a bound unit is being linked with the -R option (separate code and data).

## Defining External Symbols

A symbol can be defined as a relative location or value by specifying the LDEF or VDEF directive, respectively. The symbol's definition is then put into the symbol table by the Linker.

EDEF permits specific definitions in the Linker symbol table to be made part of the bound unit so that they are available to the Loader if the bound unit is loaded using an LDBU directive.

OVERLAYTABLE is used to put a value definition containing the name of each overlay and its overlay number in the bound unit symbol table.

COMM is used to define a labeled common block.

VAL is used to specify a value definition that is equivalent to the difference between two external location definitions.

## Protecting or Purging Symbol(s)

CPROT and CPURGE are used to protect and remove symbols associated with labeled common blocks.

PROT and UNPROTECT are used to protect and remove protection of symbols and object unit names in the symbol table. PROT prevents certain symbols and/or object unit names from being removed from the symbol table. Symbols are protected if they identify a specified address or an address within a specified range; object unit names are protected if they are equated to a specified address or an address within a specified range. UNPROTECT removes the protection from one or more items in the symbol table.

PURGE is used to remove from the symbol table unprotected symbols that define a specified address or an address within a specified range, and/or object unit names equated to a specified address or an address within a specified range.

VPURGE is used to remove a specified value definition from the symbol table.

## Reloading After System Failure

RR indicates that after a system failure a shareable bound unit can be reloaded into locations other than those it occupied at checkpoint.

## Controlling the Directive File

The user can use the -IN argument of the LINKER command to specify the user-in file from which the Linker reads directives. Otherwise, the user-in file is normally the user's terminal.



An INCLUDE directive causes the Linker to accept directives from a file specified in the directive rather than from user-in. When the Linker encounters a RETURN directive in the file specified with INCLUDE, the Linker returns to user-in.

### Terminating the Linker

QUIT is used to terminate the Linker. If a bound unit is being created, execution of the Linker terminates after the bound unit has been created. If no bound unit is being created, QUIT terminates execution of the Linker.

### LOADING THE LINKER

The command LINKER is used to load the Linker and initiate Linker processing.

After the Linker is loaded, a message is sent to the error-out file indicating the version. The message format is:

```
LINKER-rrrr-mm/dd/hhmm
```

where rrrr is a release identification, mm/dd is the month and day the Linker component was linked, and hhmm the time (hour, minutes) at which that link took place.

#### FORMAT:

```
LINKER bound-unit-path [ctl_arg]
```

#### ARGUMENTS:

bound-unit-path

Pathname of the bound unit file. The pathname can be simple, relative, or absolute and must be preceded by a space. If the specified file already exists, the existing information in the file is deleted and replaced with the new bound unit. The bound unit pathname must be specified. It may be up to 57 characters in length. The format of the bound unit file is fixed-relative.

ctl\_arg

Control arguments; none or any number of the following control arguments can be entered, in any order:

```
-IN path | -I path
```

Pathname of the disk file, card reader, operator's terminal, or other terminal through which Linker directives are entered.

When this argument is specified, the prompt character does not appear.

**-PT**

If the **-IN** argument is not specified, **-PT** can be specified to produce a prompt character on the user terminal. A prompt character is issued only if **-PT** is specified.

**-COUT list-path-name | -COUTA list-path-name**

Name of the list file. The list file can be sent to a disk, another terminal, or a printer. The **list-path-name** is associated with this list file. If **-COUT** is not specified, the **list-path-name** has a default value of **bound-unit-name.M** in the working directory. If **-COUTA** is specified, the listing is appended to the specified file.

Error messages are written to the error-out file and the list file. Linker error messages are described in the System Messages manual.

**-SIZE nn | -SZ nn**

"nn" designates the size, in units of 1024 (1K) words, of the memory area the Linker initially gets for its symbol table. "nn" must be from 2 to 64. Default is 8.

Whenever the Linker needs more symbol table space during linking, it attempts to get another memory area of nn size. This process is repeated as often as necessary.

**-W**

Save the Linker work files. Default: Linker work files are automatically released by the Linker upon Linker termination.

**-R**

Create a bound unit where all data areas defined as common are separated from all other code. Required for shareable bound units containing common data areas.

**-NOMAP**

Suppress the list file.

-SYMBOL | -SYM

Create a debugger information file. This file is used for symbolic debugging. The name of the file is `buname.V`.

-ENV string

This string must match a string at the beginning of a message library message that defines an execution environment. The bound unit is linked for the specific execution environment associated with the string (unless a `SEG` directive is used to modify the intended execution environment). In the absence of this argument, the bound unit is linked for the execution environment in which the Linker is currently running (again, unless a `SEG` directive is used to modify the intended execution environment).

Each installation is responsible for adding one or more environment-defining messages to the message library if the `-ENV` control argument is to be used. The first (or only) environment-defining message must have message number 01170. If additional environment-defining messages are needed, they must be in a message chain that begins with message number 01170. Each message describes one execution environment.

If the Linker cannot find message number 01770 or if it cannot find a match for "string" in the message chain, it continues linking the bound unit for the execution environment in which the Linker is running. The format of a message describing an execution environment is:

```
string,arg1,arg2,arg3,arg4,arg5,arg6
```

For a description of the values of `arg1` through `arg6`, refer to the `SEG` directive later in this section.

Example:

```
LINKER MYPROG -IN ^MYDISK>CNL -COUT !LPT00 -SIZE 20
```

This `LINKER` command loads the Linker and specifies the following:

- Bound unit is a fixed-relative file named `MYPROG` in the working directory.
- Linker directives are entered through disk file `^MYDISK>CNL`.

- List file goes to a line printer (configured as LPT00), rather than to a variable sequential file named MYPROG.M in the working directory.
- The Linker initially gets 20K words of memory for its symbol table.

LPT00 must have been previously defined in the DEVICE directive to the Configuration Load Manager at system generation time.

### ENTERING LINKER DIRECTIVES

Linker directives are entered through the directive input device. Most directives can also be embedded in Assembly language CTRL statements. The exceptions are noted under the descriptions of certain directives.

Linker directives consist of a directive name or a directive name followed by one or more arguments. Each directive name can be preceded by zero or more blank spaces. If one or more arguments are to be specified in a Linker directive, the directive name must be immediately followed by one or more spaces.

Multiple directives can be entered on a line by specifying a semicolon (;) after each directive except the last on the line.

The last directive on a line can be followed by a comment; to include a comment, specify a space and a slash (/) after the last directive and then enter the comment.

#### FORMAT:

```
directive [ argument ] [ argument ] [ /comment ]
```

If the directive input device is the operator's terminal or another terminal, press RETURN at the end of each line (i.e., at the end of the comment, or at the end of the last directive if there is no comment). There is no continuation between lines; the values associated with a single directive cannot be continued on a second line.

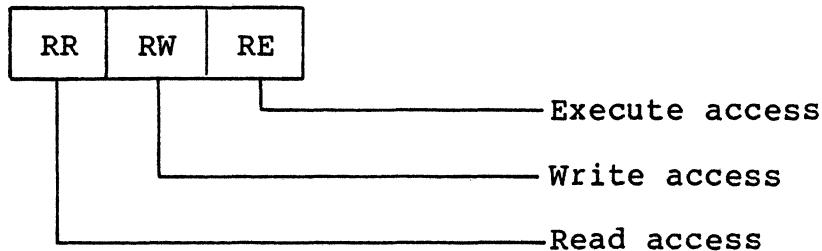
If an error occurs during the entering of a directive, an error message is written to the error-out file. Linker error messages are described in the System Messages manual. Determine what caused the error, and reenter the directive correctly. If multiple directives are entered on a line and an error occurs, the error does not affect the execution of previously designated directives. The directive that caused the error and subsequent directives on that line are not executed.

## SETTING ACCESS IN THE LINKER'S SEG OR FSEG DIRECTIVE

The method for encoding access information in the Linker's SEG or FSEG directive depends on the type of Memory Management Unit (MMU) for which the bound unit is being linked. One method is used for the Basic Memory Management Unit (BMMU) and the Extended Memory Management Unit (EMMU); a different method is used for the Virtual Memory Management Unit (VMMU). The MMU type can be ascertained by means of the Linker's REPORT directive if the user is running the Linker interactively.

### Setting Access for BMMU and EMMU Segments

Whenever access is specified in a SEG or FSEG directive for the BMMU or EMMU environment, it must be specified as a bit string of exactly 6 binary digits expressed in the form B'bbbbbb' where each b is either 0 or 1. These bit values establish read, write, and execute access for a segment according to the following pattern:

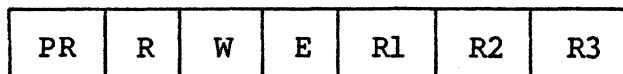


The two bit positions designating each type of access represent ring numbers. The ring numbers are coded in one's complement form: 11 = ring 0 (highest privilege); 10 = ring 1; 01 = ring 2; and 00 = ring 3 (lowest privilege). A procedure attempting to read, write, or execute a segment is given access to that segment only if the ring number at which the procedure is running is less than or equal to the ring number specified in the segment for the type of access being attempted by the procedure.

The default access value for a BMMU or EMMU segment is 000000 (i.e., ring 3 read, write, and execute access). However, if a GSHARE or SHARE directive is specified for a bound unit, the Linker changes the RW (write access) field for the root segment to 11 (i.e., ring 0 write access).

### Setting Access for VMMU Segments

Whenever access is specified in a SEG or FSEG directive for the VMMU environment, it must be specified as a bit string of from 6 to 10 binary digits expressed in the form B'b...' where each b is either 0 or 1. These bit values establish access for a segment according to the following pattern:



The meaning of each field is:

PR - Privileged Indicator (one bit). If this bit is set, privileged instructions in this segment can be executed when this segment is executed by a procedure running in ring 0. If PR = 0, a trap will occur on an attempt to execute a privileged instruction in this segment.

R - Read-permit Indicator (one bit). If this bit is set, this segment can be read by a procedure running in a ring number lower than or equal to the ring number specified in R2. If R = 0, a trap will occur on an attempt to read from this segment.

W - Write-permit Indicator (one bit). If this bit is set, this segment can be written into by a procedure running in a ring number lower than or equal to the ring number specified in R1. If W = 0, a trap will occur on an attempt to write into this segment.

E - Execute-permit Indicator (one bit). If this bit is set, instructions (unprivileged) in this segment can be executed by a procedure running in a ring number lower than or equal to the ring number specified in R2. If E = 0, a trap will occur on an attempt to execute an instruction in this segment.

Note that if E = 1 and R = 0, the segment is treated as an "execute only" segment. Such a segment is able to read its own data, despite the absence of read permission. However, the segment is not readable by other procedures.

R1 - A two-bit ring number that specifies the highest ring number (lowest privilege) from which a running procedure can write into this segment. That is, a procedure running in a ring number from 0 to R1 (the "write bracket") can write into this segment.

R2 - A two-bit ring number that specifies the highest ring number (lowest privilege) from which a running procedure can read or execute this segment. That is, a procedure running in a ring number from 0 to R2 (the "read/execute bracket") can read or execute this segment.

R3 - A two-bit ring number reserved for future use. This field should always be assigned the binary value 11.

The R1, R2, and R3 fields represent ring numbers. Each of these fields must be encoded with the actual (not complemented) binary value of the desired ring number: 00 = ring 0 (highest privilege); 01 = ring 1; 10 = ring 2; and 11 = ring 3 (lowest privilege). The system requires that the R1 ring number be less than or equal to the R2 ring number, and that R2 ring number be less than or equal to the R3 ring number. Linker enforces this rule in checking the syntax of a SEG or FSEG directive.

The default access value for a VMMU segment is 1111111111 (i.e., PR, R, W, and E bits set on, and ring 3 read, write, and execute access). However, if a GSHARE or SHARE directive is specified for a bound unit, the Linker changes the R1 value for the root segment to 00 (i.e., a ring 0 write bracket).

As mentioned above, access in a SEG or FSEG directive for a VMMU can be specified by a bit string of from 6 to 10 bits. Whenever fewer than 10 bits are specified, the Linker treats them as the low-order end of the PR-R-W-E-R1-R2-R3 field. For example, if only 6 access bits are specified, they are considered to be R1-R2-R3 ring values.

#### LINKER DIRECTIVES

Linker directives are described in alphabetic order on the following pages. Examples are provided to illustrate directive usage.

# BASE

## BASE (or BE)

Defines the relative link address within the bound unit for subsequent object units to be linked.

Unless a BASE directive specifies otherwise, the root is based at a default segment address established by the Linker, or at a segment address specified by the user (see the -ENV control argument and SEG directive). Subsequent object units are linked at successive relative addresses. A BASE directive can be used at any point during linking to change the relative locations of the root, fixed overlays, or individual object units. A BASE directive can specify a previously used or defined location, or an address relative to the beginning of the fixed area (which contains the root and any fixed overlays).

Each floatable overlay is normally based at a default segment address established by the Linker or at a segment address specified by the user (see the FSEG directive). A BASE directive in a floatable overlay can specify a previously used or defined location, or an address relative to the beginning of that same floatable overlay.

If unprotected symbols define locations that are equal to or greater than the location designated in the BASE directive, those symbols are removed from the symbol table.

The BASE directive cannot be embedded in Assembly language control statements.

### FORMAT:

|                    |   |                             |
|--------------------|---|-----------------------------|
| { BASE }<br>{ BE } | } | \$                          |
|                    |   | %                           |
|                    |   | X'address'                  |
|                    |   | =object-unit-name           |
|                    |   | xdef + X'offset'            |
|                    |   | #                           |
|                    |   | *ODD<br>*EVEN<br>*X'offset' |

### ARGUMENTS:

\$

Next location after the highest address of the linked root or just previously linked nonfloatable overlay.



## BASE

§

Highest address+1 ever used in the linked root or any previously linked nonfloatable overlay.

X'address'

A one- to five-character hexadecimal address enclosed in apostrophes and preceded by X. The specified address is relative to either the base of the fixed area (where root and fixed overlays are linked) or to the base of the current floatable overlay. For example, if root is based at a segment address of 50000, BASE X'1000' indicates a segment address of 51000. If a floatable overlay is based at a segment address of 90000, BASE X'100' indicates a segment address of 90100.

=object-unit-name

Specified object unit's base address; the subsequent root, overlay, or object unit is linked at the same relative address as the specified object unit, which must have already been linked. Furthermore, the object unit name must still exist in the symbol table (i.e., it has not been purged).

xdef + X'offset'

Any previously defined (noncommon) external symbol. If an offset is specified, it must be a hexadecimal integer with an absolute value less than  $8000_{16}$  (32,768 decimal).

#

The current address.

\*ODD

The current address, if it is odd; if it is even, base address is converted to current address+1.

\*EVEN

The current address, if it is even; if it is odd, base address is converted to current address+1.

BASE

\*X'offset'

The next location whose rightmost hexadecimal characters equal the offset (where the offset is a hexadecimal integer of four or fewer characters).

Default: \$ with the following exceptions:

The root and each floatable overlay are based at a default segment address or at a segment address specified by the user.

Example:

LINKER TEXT -COUT !LPT00 -PT  
Load Linker.

LINKER-rrrr-mm/dd/hhmm Linker identification message.

START TEXTEN Specify address where execution begins when root is loaded.

L? Linker prompt.

IST INIT Define INIT as the beginning of initialization code.

L?  
LINK OBJ1,OBJ2 Request that OBJ1.0 and OBJ2.0 be linked.

L?  
MAP Cause OBJ1.0 and OBJ2.0 to be linked, and produce a link map.

L?  
OVLY ABLE Designate end of the root, and that a nonfloatable overlay named ABLE immediately follows. The Linker assigns the number 0 to this overlay.

L?  
BASE =OBJ2

Subsequent object unit(s) constituting overlay ABLE are linked starting at the base address of the object unit OBJ2.0; this address can be determined from the map. Unprotected symbols that define locations equal to or greater than the address of OBJ2 are removed from the symbol table.

L?  
LINK OBJ5

Request that OBJ5.0 be linked.

L?  
MAP

Cause OBJ5.0 to be linked and produce a link map.

L?  
LINK OBJ6

Request that OBJ6.0 be linked.

L?  
OVLY FOX

Designate the end of the above overlay, and specify that a non-floatable overlay named FOX immediately follows. The Linker assigns the number 1 to this overlay.

L?  
BASE \$

Subsequent object unit(s) constituting the overlay named FOX are linked starting at one location higher than the ending address of OBJ6.0. This is the default BASE address, so BASE \$ need not be specified.

L?  
LINK OBJA,OBJB

Request that OBJA.0 and OBJB.0 be linked.

L?  
MAP

Cause OBJA.0 and OBJB.0 to be linked and produce a link map.

BASE

L?  
OVLY ZEBRA

Designate end of overlay 1 and name subsequent nonfloatable overlay. The Linker assigns the number 2 to this overlay.

L?  
BASE X'1105'

Designate that subsequent object units constituting overlay ZEBRA be linked based at location 1105 relative to the base of the fixed area.

L?  
LINK OBJC

Request that object unit OBJC.O be linked (based at location 1105 relative to the base of the fixed area).

L?  
LINK OBJD

Request that OBJD.O be linked.

L?  
MAP

L?  
FLOVLY FLOAT

Designate end of the above overlay, and that a floatable overlay named FLOAT immediately follows. The Linker assigns the number 3 to this overlay. This overlay is linked starting at a segment address assigned by the Linker.

L?  
LINK OBJE

Request that OBJE.O be linked.

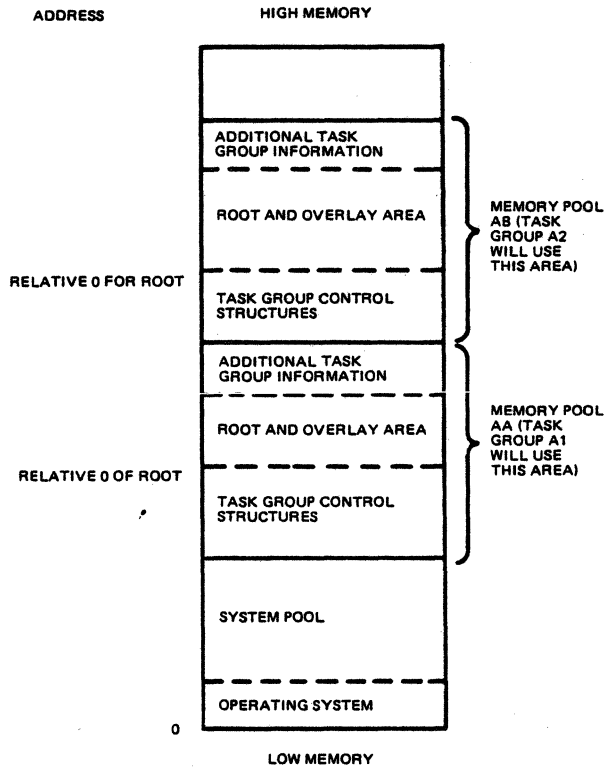
L?  
MAP

L?  
QUIT

LINK DONE  
RDY:

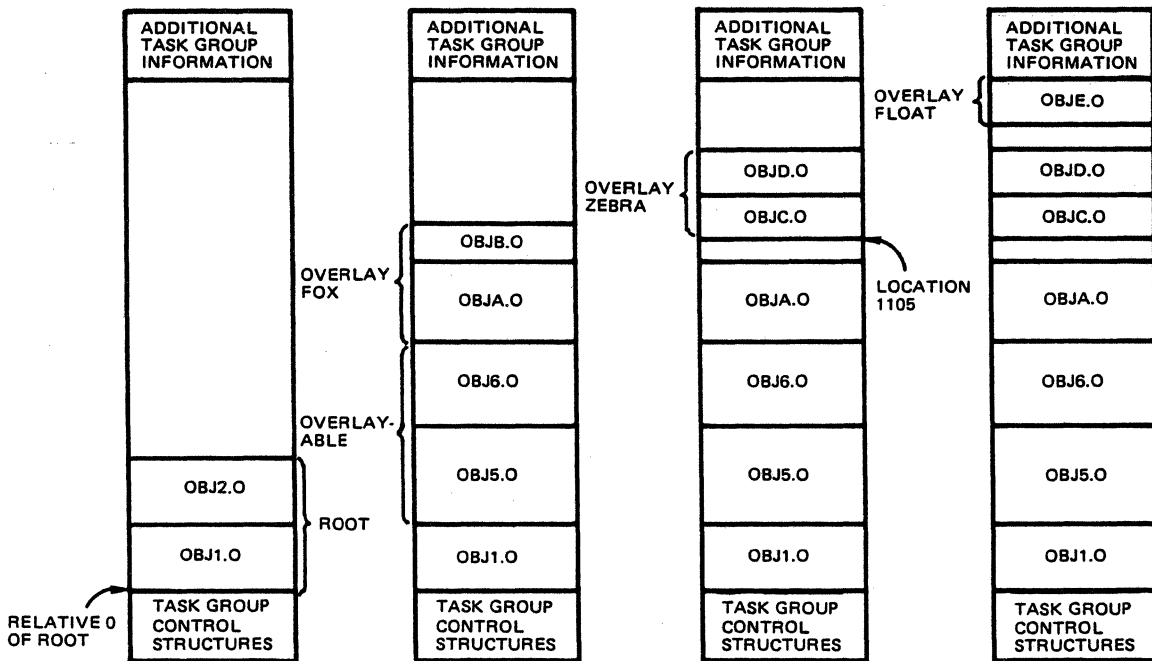
Figure 6-1 illustrates the use of BASE directives in a bound unit that consists of a root and overlays. This example assumes that the bound unit being created will be executed as part of task group A1, and memory pool AA will be used by this task group. Figure 6-1 also shows memory pool AA's location in memory relative to the system pool and another pool. The object units specified by the preceding directives are loaded into memory pool AA during execution of the bound unit.

Figure 6-2 shows the configuration of memory pool AA at different times during execution. Note that OBJ2.0 of the root is overlaid by overlay ABLE and that overlay FOX is partially overlaid by overlay ZEBRA. Also note that overlay FLOAT is positioned by the Loader and is not necessarily at the location shown in the diagram.



86-146

Figure 6-1. Relative Location of Memory in Memory Pool AA



86-147

Figure 6-2. Overlays in Memory Pool AA

## CALL CANCEL

### CALL CANCEL (CC)

Places each overlay name and its associated Linker-generated overlay number into the bound unit attribute table so that the COBOL program can call/cancel overlays by name. This directive is used when linking COBOLA programs that contain CALL/CANCEL statements to invoke overlays.

When the CC directive is specified, a special object unit, ZCCEC.O, is linked into the root to support the CALL/CANCEL facility.

The CC directive must be specified before the first LINK, LINKN, or LINKO directive in the root; it cannot be embedded in Assembly language control statements.

#### FORMAT:

CC

# COMMON

## COMMON (or COMM)

Defines a labeled "common" area of a specified size. It cannot be embedded in source code.

### FORMAT:

```
{ COMMON } symbol,X'size'  
{ COMM  }
```

### ARGUMENTS:

symbol

The external symbol to be treated as common.

X'size'

Size is specified as a one- to four-character hexadecimal number bound by single quotes and preceded by the letter X.



CPROT (or CT)

Protects the specified common symbol in the symbol table.

This directive cannot be embedded in Assembly language control statements.

FORMAT:

{ CPROT } symbol  
{ CT }

ARGUMENT:

symbol

Name of the external symbol that is to be protected. The symbol must have been specified in a COMM directive or defined as common during assembly or compilation.

# CPURGE

## CPURGE (or CE)

Removes an unprotected common symbol from the symbol table.

### FORMAT:

```
{ CPURGE } symbol  
{ CE }
```

### ARGUMENT:

symbol

The external symbol to be removed from the symbol table.  
The symbol must have been defined as common.

EDEF (or EF)

Puts the externally defined symbol(s) in the bound unit's permanent symbol table. These symbols are available for use by the Loader as described below.

When EDEF is specified, the symbol's definition must already be in the Linker symbol table.

If a bound unit is permanently resident in memory because it has been loaded at system configuration time by an LDBU directive, the definitions for any EDEFed symbols in the bound unit are placed in the system symbol table. Thereafter, whenever the Loader loads another bound unit that contains unresolved references, it tries to resolve these references by use of the definitions in the system symbol table.

If a bound unit is not permanently resident in memory, but it does contain EDEFed symbols, these symbols are used as follows:

- They can identify secondary entry points to the bound unit. Secondary entry points can be specified in the -EFN argument of the Create Group, Spawn Group, Create Task, and Spawn Task commands, as well as in the corresponding macrocalls.
- If the bound unit containing EDEFed symbols attaches or loads another subordinate bound unit (by means of \$BUAT or \$BULD macrocalls), its EDEFed symbols can be used (by the Loader) to resolve any unresolved references in the subordinate bound unit as it is loaded.

The EDEF directive can be embedded in Assembly language control statements.

## FORMAT:

```
{ EDEF } symbol1 [,symbol2]  
{ EF   }
```

## ARGUMENTS:

symbol<sub>1</sub>

Any external definition. The symbol must have been previously defined; it can name a root or overlay once the root or overlay has been linked. If the symbol was multiply defined, the first definition is used.

# EDEF

symbol<sub>2</sub>

Name of the symbol to be incorporated in the bound unit's permanent symbol table. If symbol<sub>2</sub> is not specified, the name of the symbol placed in the bound unit's permanent symbol table is that specified by symbol<sub>1</sub>.

Example:

LINKER MYPROG -PT            Load the Linker. The bound unit named MYPROG is created in the working directory. The list file MYPROG.M is also created in the working directory.

LINKER-rrrr-mm/dd/hhmm Linker identification message.

L?                            Linker prompt.

LINK A

L?  
LINKN B

L?  
MAP

L?  
EDEF BEE                      BEE is a symbol previously defined by an XDEF statement in B.O as an external location or value.

L?  
LDEF SYM,X'1234'              Assign relative location 1234 to an external symbol named SYM.

L?  
OVLY FIRST                    Declare end of root, and name non-floatable overlay that immediately follows.

L?  
LINK X,Y

L?  
EDEF SYM

L?  
QUIT

Declare that the last Linker directive has been entered. Execution of the Linker terminates after the bound unit has been created.

LINK DONE  
RDY:

LINKER PROG2 -COUT !LPT00 -PT

Load the Linker; the bound unit to be created is named PROG2. The list file is the printer. The symbol table has the default size of 8K words of memory.

LINKER-rrrr-mm/dd/hhmm Linker identification message.

L?  
LINKN W

Request that object unit W.0 be linked.

L?  
MAP

Produce a link map. In this map, it is determined that object unit W.0 contains an unresolved reference to the symbol SYM, which was defined in the root of the bound unit MYPROG.

L?  
QUIT

LINK DONE  
RDY:

If MYPROG is loaded into memory by an LDBU configuration directive, when the Loader loads PROG2 the Loader resolves the unresolved reference in PROG2 to the symbol SYM, which was defined in the root of MYPROG.

An EDEF directive cannot be entered on the same directive line that causes linking of the object unit that defines the EDEF's symbol. For example, if the symbol TAG is defined in object unit A, the following directive line is not allowed:

LINK A;EDEF TAG.

# FLOATB6

## FLOATB6 (or F6)

Suppresses certain error checking on local common references when the -R argument has not been used. The directive tells the Linker that the user manages \$B6 and causes each local common reference to be relocated as if the \$B6 pointed to the base of the fixed or floatable overlay containing the reference. (If an overlay is equal to or greater than 32K words, each local common reference is relocated as if \$B6 pointed to the base of the overlay plus 32K words.) Normally, \$B6 is set by the system to the base of the fixed (root and fixed overlay) area, and \$B6-type local common references within floatable overlays would be invalid.

Before using this directive, consult the person responsible for system building and determine available system memory.

This directive must be specified before the first object unit containing a local common reference is linked.

### FORMAT:

```
{ FLOATB6 }  
{ F6 }
```

FLOVLY (or FY)

Assigns the specified name and a number to the floatable overlay that immediately follows, and designate the end of the preceding root or overlay. The characteristics of floatable overlays are described at the end of this directive description.

FLOVLY must be specified as the first directive of each floatable overlay.

The Linker assigns a four-digit number to each overlay. Overlays are numbered sequentially in ascending order; the first overlay is 0.

FORMAT:

{ FLOVLY } name  
 { FY }

ARGUMENT:

name

Name of the floatable overlay that immediately follows. For character conventions, see "OVLY" later in this section.

Example:

LINKER BU -PT                    Load the Linker and designate BU as the bound unit name.

LINKER-rrrr-mm/dd/hhmm    Linker identification message.

L?  
 LINK A,B

L?  
 MAP                            Produce a link map.

L?  
 FLOVLY GR                    Declare the end of the root that consists of object units A.0 and B.0, and specify that the next overlay is a floatable overlay named GR. The Linker assigns the number 0 to this overlay.

## FLOVLY

L?  
LINK X,Y; MAP  
L?  
FLOVLY BR

Declare the end of floatable overlay GR and designate that the floatable overlay that immediately follows is BR. The Linker assigns the number 1 to this overlay.

L?  
LINK R6

L?  
MAP

L?  
QUIT

LINK DONE  
RDY:

External location definitions defined within a floatable overlay are automatically purged at the end of the overlay because they cannot be referred to from outside the overlay.

A floatable overlay must have the following characteristics:

- External location definitions in the overlay are not referred to from the root or any other overlay.
- There cannot be external references between floatable overlays.
- The overlay must be linked after all desired nonfloatable overlays have been linked.
- The overlay cannot contain P+DSP references to any other overlay or to the root.
- The overlay can contain IMA (immediate memory address) references to locations within itself and/or IMA references to the root and fixed overlays.



FSEG (or FG)

Define the execution environment in which a floatable overlay will run. Before using this directive, consult with the person responsible for system building and determine the segment numbers available to task groups. If used, the FSEG directive must appear after a FLOVLY directive and before the first LINK/LINKN/LINKO, BASE, LDEF, PROTECT, PURGE, or UNPROTECT directive for the floatable overlay. With this directive, you can specify the segment number to be assigned, as well as the access rights to the segment. If FSEG is not used, the Linker assigns a base segment number and default access rights to the floatable overlay.

It is generally best to let the Linker assign segment numbers for floatable overlays. Since floatable overlays must be linked after the root and all fixed overlays, the Linker can determine which segment numbers have already been used when it is ready to link the first floatable overlay. If you don't specify a segment number for the first floatable overlay, the Linker assigns one as follows:

- If the bound unit doesn't contain separated data (i.e., linked without the -R control argument), the Linker assigns the first floatable overlay a segment number one higher than the highest number used for the fixed area (root and fixed overlays). The Linker then increases the segment number for each successive floatable overlay.
- If the bound unit contains separated data, the Linker assigns the first floatable overlay a segment number one higher than the number assigned to the separated data (either by default or explicitly by the user). If you know that the separated data requires more than one segment, you can use the FSEG directive to assign the first floatable overlay a segment number one higher than the highest number expected to be used for the data.

## FORMAT:

```
{ FSEG } arg1, arg2
{ FG   }
```

## FSEG

### ARGUMENTS:

#### arg1

Specifies the base segment number to be used for the floatable overlay. The value must be a hexadecimal number from 1 to F for a BMMU environment, from 8C to FF for an EMMU environment, and from 3B to 3FF for a VMMU environment.

Segment numbers must be specified in hexadecimal notation; e.g., X'h...'.

#### arg2

Specifies the access rights for the floatable overlay. The value must be a bit string of 6 binary digits for the BMMU or EMMU environment and from 6 to 10 binary digits for the VMMU environment. The value must be specified in binary string notation; e.g., B'b...'. The bit string represents the corresponding access fields in the segment descriptor. For more information on setting access, refer to "Setting Access in the Linker's SEG or FSEG Directives" earlier in this section.

#### Example 1:

```
FSEG X'9'
```

In this example, segment 9 with default access is assigned to the floatable overlay.

#### Example 2:

```
FSEG X'90',B'001000'
```

In this example, segment 90 with ring 1 write access is assigned to the floatable overlay.

## GSHARE (or GE)

Indicates that the bound unit is globally shareable, which means that the program is shareable among groups, and the root is always loaded into the system memory pool. This directive should not be used if a SHARE directive would suffice. System performance can be affected if this directive is misused. Floatable overlays are loaded into user space and are not shareable unless overlay area tables (OATs) are used.

GSHARE causes the root of the bound unit to be assigned ring 0 write access. If the globally shareable bound unit contains any floatable overlays or a separate data area, the Linker leaves them with the default write access (ring 3) or with any user-supplied write access. Before using this directive, consult with the person responsible for system building and determine available system memory.

Nonsharable bound units (linked without SHARE or GSHARE) are always loaded into the user's memory pool.

### FORMAT:

```
{ GSHARE }  
{ GE }
```

## IN

### IN

Change the primary directory. The primary directory is the first one the Linker searches for the specified object unit(s) to be linked. The default primary directory is the working directory.

The IN directive must be specified before the first LINK, LINKN, or LINKO directive that requests the linking of an object unit that is in the specified directory.

The specified directory remains the primary directory until another IN directive is entered. If the primary directory is changed by an IN directive and at a later time you want the task group's working directory to be the primary directory, enter the IN directive and omit the pathname.

The IN directive cannot be embedded in Assembly language control statements.

#### FORMAT:

IN [path]

#### ARGUMENT:

path

Pathname of the directory being designated as the primary directory. The pathname can contain a maximum of 57 characters. A simple, relative, or absolute pathname can be specified. (Methods of designating pathnames are described in Section 3 of this manual). If the path is omitted, the working directory becomes the primary directory.

#### Example 1:

IN ^DIR>PRIM

This directive designates that ^DIR>PRIM is the primary directory.

## Example 2:

This example illustrates use of the IN directive in conjunction with directives that request the linking of object units. Assume that the primary directory is the working directory, whose absolute pathname is ^WORK>CURR; object units X.O and Y.O are in the working directory. A.O and C.O are not in the working directory.

|                        |   |
|------------------------|---|
| LINKER OUTPUT -PT      | Load the Linker; a bound unit named OUTPUT is created on the working directory.   |
| LINKER-rrrr-mm/dd/hhmm | Linker identification message.  |
| L?<br>LINKN X          | Request the linking of object unit X.O; X.O is in the working directory.  |
| L?<br>IN ^NEW>PRIM     | Designate ^NEW>PRIM as the primary directory.   |
| L?<br>LINKN A,C        | Request the linking of object units A.O and C.O in the primary directory. ^NEW>PRIM>A.O is the pathname of A.O and ^NEW>PRIM>C.O is the pathname of C.O, as expanded by the Linker. |
| L?<br>IN               | Designate the working directory as the primary directory.   |
| L?<br>LINKN Y          | Request the linking of object unit Y.O, in the working directory. ^WORK>CURR>Y.O is the pathname of Y.O, as expanded by the Linker.   |
| L?<br>MAP; QUIT        |   |
| LINK DONE<br>RDY:      |   |

# INCLUDE

## INCLUDE (or IE)

Accept directives from a file other than user-in or the file specified in the -IN argument of the LINKER command. When the Linker encounters an end of file or a RETURN directive in the file specified by the INCLUDE directive, it again seeks directives from the previously active file. If used, the INCLUDE directive must be the last directive entered on a line.

The directive file specified by the INCLUDE directive cannot contain an INCLUDE directive.

The INCLUDE directive cannot be embedded in Assembly language control statements.

### FORMAT:

```
{ INCLUDE } path  
{ IE }
```

### ARGUMENT:

path

Pathname of the file from which the Linker directives are to be read. A simple pathname can be up to 12 characters in length; an absolute pathname can be up to 57 characters in length.

### Example:

```
INCLUDE NEW
```

This directive causes the Linker to accept directives from a file named NEW in the working directory.

INIT2 (or I2)

Specifies a 2-word initialization pattern for all otherwise uninitialized common blocks if code and data are separated. If code and data are separated and the INIT2 directive is not used, all otherwise uninitialized common blocks are initialized to null by the the Loader as it loads the separated data load unit.

## FORMAT:

```
{ INIT2 } X'hhhhhhh'
```

```
{ I2 }
```

## ARGUMENT:

X'hhhhhhh'

Hexadecimal initialization pattern comprising exactly 8 integers enclosed in apostrophes and preceded by X.

## Example:

```
INIT2 X'5555555'
```

In this example, assuming code and data are separated, all otherwise uninitialized common blocks are initialized by the Loader to the specified value (ASCII Us).

# IST

## IST (or IT)

Identifies the initialization code start address in the root. Initialization code is to be executed once, immediately after the root is loaded at system boot time. After the initialization code is executed, its space can be made available for overlays. The IST directive can be used only with a bound unit that contains an initialization subroutine table and is loaded at system configuration time by means of an LDBU directive. LDBU, a CLM directive, is explained in the System Building and Administration manual. IST is not meaningful unless the bound unit is specified in an LDBU directive.

The IST directive cannot be embedded in Assembly language control statements.

### FORMAT:

{ IST } external symbol  
{ IT }

### ARGUMENT:

external symbol

Symbol identifying the beginning of the IST section of the bound unit.



LDEF (or LF)

Defines an external symbol and assigns it a relative location. A symbol should be defined only once, either as a location or as a value. When a symbol is defined, its definition is put into the Linker symbol table so that it can be used to resolve references to the symbol during linking. When a symbol defined as a location is no longer used, its symbol table entry can be cleared by specifying the PURGE directive. PURGE has no effect if a PROTECT (PROT) directive was previously specified; however, a protected symbol can be named in an UNPROTECT directive and then in a PURGE directive.

The LDEF directive cannot be embedded in Assembly language control statements.

## FORMAT:

$$\left. \begin{array}{l} \{ \text{LDEF} \} \\ \{ \text{LF} \} \end{array} \right\} \text{symbol, } \left( \begin{array}{l} \$ \\ \% \\ \text{X'address'} \\ =\text{object-unit-name} \\ \text{xdef } \{ \_ \} \text{X'offset'} \\ \# \end{array} \right)$$

## ARGUMENTS:

symbol

The symbol can include any character that is legitimate for a file name; see the LINK directive.

\$

Next location after the highest address of the linked root or just previously linked nonfloatable overlay.

%

Highest address+1 ever used in the linked root or any previously linked nonfloatable overlay.

## LDEF

### X'address'

A one- to five-character hexadecimal address enclosed in apostrophes and preceded by X. The specified address is relative to either the base of the fixed area (where root and fixed overlays are linked) or to the base of the current floatable overlay. For example, if root is based at a segment address of 50000, LDEF LLEWELLYN,X'1234' indicates a segment address of 51234.

### =object-unit-name

Specified object unit's base address.

### xdef[ + X'offset']

Any previously defined external symbol. If an offset is specified, it must be a hexadecimal integer with an absolute value less than  $8000_{16}$  (32,768 decimal).

#

The current address.

### Example:

|                         |  |
|-------------------------|--|
| LINKER BOUND -PT        | Load the Linker and designate BOUND as the bound unit name.        |
| LINKER-rrrr-mm/dd/hhmm  | Linker identification message.                                     |
| L?<br>LINK A, B, C      |  |
| L?<br>MAP               |  |
| L?<br>LDEF SYM, X'1234' | SYM assigned location 1234 relative to the base of the fixed area. |
| L?<br>OVLY FIRST        | Declare end of root and name first nonfloatable overlay.           |
| L?<br>LINK R; MAP       |  |

L?  
LDEF QUIZ,=C

QUIZ assigned base location of the previously linked object unit named C.O.

L?  
OVLY SECOND

L?  
LINKN D; LINK F; MAP

L?  
LDEF NEW,SYM

NEW assigned same location as the symbol SYM, which was defined in the root; i.e., NEW is assigned location 1234 relative to the base of the fixed area.

L?  
OVLY NEXT

L?  
BASE X'1300'

This overlay is based at location 1300 relative to the base of the fixed area.

L?  
LINK W,X; MAP

L?  
LDEF ANY,\$

ANY assigned next location after highest address of the previously linked nonfloatable overlay, SECOND.

L?  
OVLY THIRD

L?  
LINK Z

L?  
LINK Q; MAP

LDEF

L?

LDEF FIND,%

FIND assigned next location after highest address of the root or any previously linked nonfloatable overlay. (A previous nonfloatable overlay was named SECOND; if it ended at location 1566 relative to the base of the fixed area and if this is the highest location reached during the linking of object units constituting this bound unit, FIND is assigned location 1567 relative to the base of the fixed area.)

L?

QUIT

LINK DONE

RDY:

This example illustrates the use of each format of the LDEF directive.

LIB (or LIB1)

Designate a directory as the secondary directory. This directive permits the linking of object units that are in directories other than the primary directory. If an object unit specified in the LINK, LINKN, or LINKO directive cannot be found in the primary directory, the Linker searches the secondary directory.

If LIB is not specified, there is no secondary directory; the Linker searches only the primary directory.

The specified secondary directory remains in effect until the LIB directive is respecified with a different directory name, or without any directory name.

All specified object units in the primary directory are linked first; then all specified object units in the secondary directory are linked, and so on. To cause object units to be linked in an order that is independent of their location, the LINKN or LINKO directive must be used.

The LIB directive must be specified before the first LINK, LINKN, or LINKO directive that requests the linking of an object unit in the secondary directory.

## FORMAT:

LIB [path]

## ARGUMENT:

path

Pathname of the directory being designated as the secondary directory. A simple, relative, or absolute pathname can be specified. (Methods of specifying pathnames are described in Section 3.) If path is omitted, a previously specified directory is removed from the list of directories to be searched by the Linker.

## Example 1:

LIB DIR&gt;SECND

This directive designates DIR>SECND as the relative pathname of the secondary directory.

Example 2:

LIB DIR>SECND

Designate DIR>SECND as the relative  
pathname of the secondary directory.

LINK B

Request the linking of object unit B.O;  
B.O resides in the primary directory.

LINK A

Request the linking of object unit A.O;  
A.O resides in the primary directory.

LINK W

Request the linking of object unit W.O;  
W.O resides in the secondary directory.  
DIR>SECND>W.O is the relative pathname of  
W.O, as expanded by the Linker.

## LIB2, LIB3, OR LIB4

### LIB2, LIB3, OR LIB4

Designates directories as the third, fourth, or fifth directory. If an object unit specified in the Linker directive cannot be found in the primary or secondary directory, then the third directory is searched and so on.

The specified directories remain in effect until another LIB2, LIB3, LIB4 statement is given.

The LIB2, LIB3, or LIB4 directive must be specified before the first LINK, LINKN, or LINKO directive that requests the linking of an object unit in one of these directories.

#### FORMAT:

{ LIB2 }  
{ LIB3 } [path]  
{ LIB4 }

#### ARGUMENT:

path

Pathname of the third, fourth, or fifth directory to be searched (if LIB is specified) if the object unit specified in a Linker directive is not found in the preceding directories. A simple, relative, or absolute pathname can be specified. If path is omitted, a previously specified directory (2, 3, or 4) is removed from the list of directories to be searched by the Linker.

# LINK

## LINK (or LK)

Links one or more specified object units. Each specified object unit name is put into the link request list. The object units are linked when the first subsequent directive other than LINK or START is encountered. When this occurs, the Linker searches the primary directory and links the specified object units in the primary directory in the order in which they were requested. If all of the object units are not found and there is a secondary directory, the Linker searches the secondary directory and links specified object units found there, in the order in which they were requested. If there is a copy of an object unit in both the primary and secondary directory, the copy in the primary directory is linked.

The order in which object units are linked is important for the following reasons: (1) it determines which object units are in memory when parts of the root or overlay are overlaid, and (2) within the root and each overlay, the first start address encountered by the Linker (either in an END statement or a START directive) is used as the start address for that root or overlay.

During each execution of the Linker, at least one LINK, LINKN, or LINKO directive must be entered for each root or overlay. Multiple LINK directives can be specified within a single root or overlay. If LINK and/or LINKN and/or LINKO directives request that the same object unit be linked more than once within a single bound unit, only the first request is honored, unless the object unit name has been purged.

LINK directives can be embedded in Assembly language control statements; the specified object unit(s) are added to the end of the current link request list. See "LINKN" and "LINKO" later in this section for the order in which object units are linked if there are embedded LINK directives and/or LINKN and/or LINKO directives.

### FORMAT:

```
{ LINK } obj-unit1 [,obj-unit2] ...  
{ LK }
```

### ARGUMENTS:

obj-unit

Name of an object unit to be linked. The name of each object unit must conform to the conventions for specifying disk file names; see Section 3 of this manual for file name conventions.



LINKN (or LN)

Link object units in the exact order specified.

If directives request that an object unit be linked more than once within a single bound unit, only the first request is honored, unless the object unit name has been purged.

During each execution of the Linker, at least one LINKN, LINK, or LINKO directive must be specified for each root or overlay.

Multiple LINKN directives can be specified within a single root or overlay.

LINKN directives can be embedded in Assembly language control statements; the specified object unit(s) are added to the end of the link request list and the library search restarts at the primary directory.

## FORMAT:

```
{ LINKN } obj-unit1 [,obj-unit2] ...
{ LN }
```

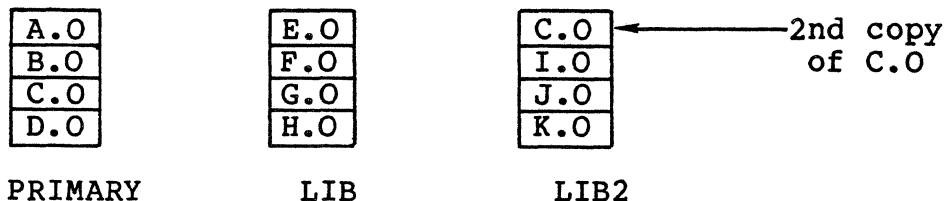
## ARGUMENTS:

obj-unit

Name of an object unit to be linked. See the description of the obj-unit under "LINK."

## Examples:

In the following examples, assume that the working directory is the primary directory and LIB and LIB2 directives have been specified.



Note that two copies of module C.O have been created. Throughout the following examples, the first (primary directory) copy is designated C(1) and the second (LIB2) copy is designated C(2).

## LINKN

Example 1:

```
LINK A,G,K,C,F
```

The modules are linked in the following order:

```
A,C(1),G,F,K
```

Example 2:

```
LINKN A,G,K,C,F
```

The modules are linked in the following order:

```
A,G,K,C(1),F
```

Example 3:

```
LINK A,G,K,C,F
```

Assume that module G.O contains CTRL LINK B,J. The modules are be linked as follows:

```
A,C(1),G,F,K,B,J
```

Once the Linker has started to search LIB, it does not return to the primary directory unless a new link request list is found. The two embedded requests are added to a new link request list, forcing a rescan of all libraries.

Example 4:

```
LINKN A,G,K,C,F
```

Assume that module G.O contains CTRL LINKN B,J. The modules are linked as follows:

```
A,G,K,C(1),F,B,J
```

Example 5:

```
LINKN G,B
```

Assume that module G.O contains CTRL LINK C. The modules are linked as follows: G,B,C(1)

Example 6:

LINK G,D,F

Assume that module G.O contains CTRL LINK C,B. The modules are linked as follows:

D,G,F,C(1),B

Example 7:

LINK G,D,F

Assume that module G.O contains CTRL LINKN C,B. The modules are linked as follows:

D,G,F,C(1),B

## LINKnn

### LINKnn

Link the specified object unit(s) if bit nn is turned on. This directive allows selective linking.

The LINKnn directive must be used in conjunction with the VDEF directive (or a VALDEF directive in a compilation unit). The VDEF directive is used to modify the bit setting in a 32-bit array. The leftmost 16 bits in the array are set by the symbol Z\_MSKR; the rightmost 16 bits in the array are set by the symbol Z\_MSKU. Through the VDEF directive, you assign a value to Z\_MSKR or Z\_MSKU that sets the appropriate bit ON (a value of 1) or OFF (a value of 0).

Each occurrence of LINKnn causes the array to be indexed by nn. If the referenced bit is ON (1), the link request is processed. If the referenced bit is OFF (0), the link request is ignored.

The bits in the array are initially set ON; i.e., all LINKnn directives are processed. The array is modified by the VDEF directive (as described above). The VPURGE directive must be used to remove Z\_MSKR and Z\_MSKU from the symbol table before these symbols can be redefined.

#### FORMAT:

LINKnn obj-unit<sub>1</sub> [,obj-unit<sub>2</sub>...]

#### ARGUMENTS:

nn

Two-digit hexadecimal value between 00 and 1F used as an index in a 32-bit array.

obj-unit

Name of the object unit to be linked if bit nn of the 32-bit array is ON. See the description of obj-unit under "LINK".

## LINKO

### LINKO (or LO)

Operates in the same manner as the LINKN directive, except that all embedded Linker directives in the named object units are ignored.

Only the named object units are linked.

The LINKO directive cannot be embedded in Assembly language control statements.

#### FORMAT:

```
{ LINKO } obj-unit1 [,obj-unit2] ...  
{ LO   }
```

#### ARGUMENT:

obj-unit

Name of an object unit to be linked. See the description of obj-unit under "LINK."

# LSR

## LSR

List the Linker search rules. The directories to be searched by the Linker for object unit(s) are listed in the order in which they will be searched.

FORMAT:

LSR

## MAP, MAPD, AND MAPU

### MAP, MAPD, and MAPU (OR MP, MD, and MU)

Creates a link map containing: (1) externally defined symbols that have not been purged, and (2) any unresolved references to external symbols. The link map is written to the list-file (see -COUT in the Linker command).

If MAP or MAPD is specified, each externally defined symbol and unresolved reference generated by the linking of object units is listed in the map and preceded by the name of the object unit in which it is located. A map also includes the names of object units that were linked because of embedded Linker directives, and the externally defined symbols and unresolved references contained in those object units. If the MAP directive immediately precedes a QUIT directive, the link map contains all the unpurged externally defined symbols and unresolved references of the completed bound unit.

The MAPD directive is meant to be used if a bound unit is being linked with the -R option (separated code and data). Although MAPD can be specified anywhere among the Linker directives, it has a delayed effect. It causes the Linker to produce a link map after the QUIT directive has been encountered and at a time when all common blocks in the separate data area have been assigned addresses. (If a bound unit is linked with separated code and data, each common block is listed with its current size, rather than its address, in any map that appears before the QUIT directive.)

The MAPU directive lists only unresolved references. If MAPU is specified, the map contains each unresolved reference and the object unit in which it is located.

Any of the map directives can be interspersed among other Linker directives. When these directives are encountered, all object units named in the link request list are linked before a map is produced. Maps are useful for determining whether all required object units have been linked, and whether all symbols referred to in those object units have been defined.

If there are any unresolved references remaining after the last object unit is linked, a MAPU directive is automatically generated by the Linker.

# MAP, MAPD, AND MAPU

## FORMAT:

( MAP  
MP  
MAPD  
MD  
MAPU  
MU )

Default: No map produced.

A full link map (a map generated by the MAP directive) comprises the following sections:

|                       |  |
|-----------------------|--|
| Start                 | Address at which execution of the current load unit (root or overlay) begins; specified in the START directive or in a linked object unit.   |
| Low                   | Memory address at which the current load unit is based.  |
| High                  | Next location after the highest address of the current load unit.  |
| \$LCOMW               | Address or size assigned to local common for the current load unit. If no local common is defined for the current load unit, this does not appear on the MAP.  |
| Current               | Next location after the current address of the current load unit (when the map was created).   |
| External Definitions  | All external symbols currently defined in the symbol table. Unprotected symbols defined in the root or a previously linked overlay appear in the map unless the symbols are purged by a PURGE or BASE directive. Symbols erroneously defined as both a value and a location appear twice under External Definitions. |
| Unresolved References | All references to undefined symbols contained in the root and overlay(s) are listed in the map.  |

For the root and each overlay containing unresolved references, the following information is presented:

- Root and overlay(s) containing references to undefined symbol(s)



- Relative address of the last reference to the symbol

If an undefined symbol is referred to in multiple overlays, the symbol is listed in the map more than once.

If there are external references in both P-relative and Immediate Memory Address forms to an undefined symbol, the symbol is listed twice under Unresolved References.

External symbol names and common block names can contain up to 127 characters. Because of the potentially great variation in the length of these names, two different formats are possible for the list of common blocks or externally defined symbols that appear under a given object unit name in the link map. If an object unit contains no names greater than 10 characters, the portion of the link map pertaining to that object unit appears in a three-column format. Otherwise, the portion of the link map pertaining to that object unit appears in a two-column format, and names that exceed 23 characters are truncated at that point. Figure 6-3 illustrates the use of the MAP and MAPU directives.

The date and time at which the bound unit was created is automatically put in the bound unit's header area.

# MAP, MAPD, AND MAPU

EXMPL 1986/09/09 1354:43.7 Page 1

LINKER-(Linker identification) (System identification)  
Bu= EXMPL Linked on: 1986/09/09 1354:43.7 -R -SYM

-> LIB >>LDD>ZF1RT

-> IN M4LNKR

-> LIB2 M4LNKR>TESTPROGS>ASSEMBLER

-> VDEF Z\_MSKR,X'4000'

-> BASE X'10'

-> START START1

-> LINK RTPROG

-> MAP

M4LNKR>TESTPROGS>ASSEMBLER>RTPROG.O

RTPROG (00020010)

1986/01/09 1353:17.7 MAP-1.1 -08/04/1008 GCOS6 MOD400-R3.1-06/11/1913 PAGE

[ EDEF START1]

[ EDEF OLI1]

[ EDEF OLI2]

[ EDEF OLI3]

Figure 6-3. Link Map Formats

```

EXMPL                1986/09/09 1354:43.7                Page 2

*****
*****      M A P      *****
*****

**Start:    0002001E
**Low:      00020000
**High:     00020042
**Current:  00020042

***** Common Block Definitions *****

** EXMPL 00020000
* RTPROG 00020010
  C COMM1    00000032    C COMM2    00000064    LX LCOMM1    00000032
  LX $LCOMW  00000064

***** External Derinitions *****

      P ZHCOMM    00000000    P ZHREL    00020000    V Z_MSQR    00004000

** EXMPL 00020000
* RTPROG 00020010
  START1    0002001E    ST2    00020011    ST3    00020013
  ST4    00020015    ERP    00020017    OLI1    00020010
  OLI2    00020025    OLI3    0002002E    OLI4    00020037

***** Unresolved References *****

** EXMPL 00020000
* RTPROG 00020010
  V OVLAY1    0002001F    V OVLAY2    0002002A    V OVLAY3    00020033
  V OVLAY4    0002003C

Key: **=Root or overlay name, or heading; *=Object file name; C=Common;
      L=Local common; D=Displacement reference; V=Value; P=Protected; X=Purged;

*****
*****
*****

-> OVLY OVLAY1

-> LINKN RTPROG.00

M4LNKR>TESTPROGS>ASSEMBLER>RTPROG.00.0
OVLAY1 (00020042)
1986/01/09 1338:32.8 MAP-1.1 -08/04/1008 GCOS6 MOD400-R3.1-06/11/1913 PAGE

-> LINK01 TIME

-> LINK02 WAIT

-> LDEF ST4A,ST4+X'10'

>>LDD>ZF1RT>TIME.O
TIME 8310260 (00020056)

```

Figure 6-3 (cont). Link Map Formats

MAP, MAPD, AND MAPU

```
EXMPL          1986/09/09 1354:43.7                      Page 3

1983/10/26 1423:11.9 MAP-1.1 -08/04/1008 GCOS6 MOD400-R3.0-09/17/1626 PAGE
[ LINK Z1TIME]
>>LDD>ZF1RT>Z1TIME.O
Z1TIME 8310270 (00020056)
1983/10/27 1038:27.2 MAP-1.1 -08/04/1008 GCOS6 MOD400-R3.0-09/17/1626 PAGE
-> EDEF ST3
-> MAP
```

Figure 6-3 (cont). Link Map Formats

EXMPL 1986/09/09 1354:43.7

Page 4

\*\*\*\*\*  
 \*\*\*\*\* M A P \*\*\*\*\*  
 \*\*\*\*\*

\*\*Start: 00020048  
 \*\*Low: 00020042  
 \*\*High: 0002007B  
 \*\*Current: 0002007B

\*\*\*\*\* Common Block Definitions \*\*\*\*\*

\*\* EXMPL 00020000  
 \* RTPROG 00020010  
   C COMM1 00000032 C COMM2 00000064 LX LCOMM1 00000032  
   LX \$LCOMW 00000064  
 \*\* OVLAY1 00020042  
 \* RTPROG.00 00020042  
   LX LCOMM1 00000032 LX \$LCOMW 00000096 LX LCOMM2 00000032

\*\*\*\*\* External Derinitions \*\*\*\*\*

P ZHCOMM 00000000 P ZHREL 00020000 V Z\_MSKR 00004000  
 \*\* EXMPL 00020000  
 \* RTPROG 00020010  
   START1 0302001E ST2 00020011 ST3 00020013  
   ST4 00020015 ERP 00020017 OLI1 00020010  
   OLI2 00020025 OLI3 0002002E OLI4 00020037  
 V OVLAY1 00000001  
 \*\* OVLAY1 00020042  
 \* RTPROG.00 00020042  
 \* TIME 00020056  
 \* Z1TIME 00020056  
   TIME 00020056 ST4A 00020025

\*\*\*\*\* Unresolved References \*\*\*\*\*

\*\* EXMPL 00020000  
 \* RTPROG 00020010  
   V OVLAY2 0002002A V OVLAY3 00020033 V OVLAY4 0002003C

Key: \*\*=Root or overlay name, or heading; \*=Object file name; C=Common;  
 L=Local common; D=Displacement reference; V=Value; P=Protected; X=Purged;

\*\*\*\*\*  
 \*\*\*\*\*  
 \*\*\*\*\*

-> MAPU

Figure 6-3 (cont). Link Map Formats

MAP, MAPD, AND MAPU

```
EXMPL                1986/09/09 1354:43.7                Page 5

* * * * *
* * * * *   M A P   * * * * *
* * * * *

**Start:    00020048
**Low:      00020042
**High:     0002007B
**Current:  0002007B

***** Unresolved References *****

** EXMPL 00020000
* RTPROG 00020010
  V OVLAY2    0002002A    V OVLAY3    00020033    V OVLAY4    0002003C

Key: **=Root or overlay name, or heading; *=Object file name; C=Common;
      L=Local common; D=Displacement reference; V=Value; P=Protected; X=Purged;

* * * * *
* * * * *
* * * * *

-> PROTECT OLI4

-> PURGE ST2

-> BASE OLI3

-> OVLY OVLAY2

-> LINKO RTPROG.01

^M4LNKR>TESTPROGS>ASSEMBLER>RTPROG.01.0
OVLAY2      8310260      (0002002E)
1986/01/09 1339:30.0 MAP-1.1 -08/04/1008 GCOS6 MOD400-R3.1-06/11/1913 PAGE

-> LINK01 WAIT

-> MAP

>>LDD>ZF1RT>WAIT.O
WAIT        8310260      (00020042)
1983/10/26 1424:02.9 MAP-1.1 -08/04/1008 GCOS6 MOD400-R3.0-09/17/1626 PAGE

[ LINK Z1WAIT]

>>LDD>ZF1RT>Z1WAIT.O
Z1WAIT      8310270      (00020042)
1983/10/27 1116:12.2 MAP-1.1 -08/04/1008 GCOS6 MOD400-R3.0-09/17/1626 PAGE
```

Figure 6-3 (cont). Link Map Formats

```

EXMPL                1986/09/09 1354:43.7                Page 6

*****
*****      M A P      *****
*****

**Start:    00020034
**Low:      0002002E
**High:     0002005B
**Current:  0002005B

***** Common Block Derinitions *****

** EXMPL 00020000
* RTPROG 00020010
  C COMM1    00000032    C COMM2    00000064    LX LCOMM1    00000032
  LX $LCOMW  00000064

** OVLAY1 00020042
* RTPROG.00 00020042
  LX LCOMM1    00000032    LX $LCOMW    00000096    LX LCOMM2    00000032

** OVLAY2 0002002E
* RTPROG.01 0002002E
  LX LCOMM1    00000032    LX $LCOMW    000000C8    LX LCOMM2    00000032

***** External Definitions *****

  P ZHCOMM    00000000    P ZHREL    00020000    V Z_MSKR    00004000

** EXMPL 00020000
* RTPROG 00020010
  START1    0002001E    X ST2    00020011    ST3    00020013
  ST4    00020015    ERP    00020017    OLI1    00020010
  OLI2    00020025    X OLI3    0002002E    P OLI4    00020037
  V OVLAY1    00000001

** OVLAY1 00020042
* RTPROG.00 00020042
* TIME 00020056
* Z1TIME 00020056
  X TIME    00020056    ST4A    00020025    V OVLAY2    00000002

** OVLAY2 0002002E
* RTPROG.01 0002002E
* WAIT 00020042
* Z1WAIT 00020042
  WAIT    00020042

***** Unresolved References *****

** EXMPL 00020000
* RTPROG 00020010
  V OVLAY3    00020033    V OVLAY4    0002003C

Key: **=Root or overlay name, or heading; *=Object file name; C=Common;
      L=Local common; D=Displacement reference; V=Value; P=Protected; X=Purged;
  
```

Figure 6-3 (cont). Link Map Formats

MAP, MAPD, AND MAPU

```
EXMPL          1986/09/09 1354:43.7                      Page 7

*****
*****
*****

-> LSR

*****
PRIM  M4LNKR
LIB   >>LDD>ZF1RT
LIB2  M4LNKR>TESTPROGS>ASSEMBLER
*****

-> Q
```

```
EXMPL          1986/01/09 1354:43.7                      Page 8

*****
*****      M A P      *****
*****

**Start:      00020034
**Low:        0002002E
**High:       0002005B
**Current:    0002005B

***** Unresolved References *****

** EXMPL 00020000
* RTPROG 00020010
  V OVLAY3      00020033      V OVLAY4      0002003C

Key: **=Root or overlay name, or heading; *=Object file name; C=Common;
      L=Local common; D=Displacement reference; V=Value; P=Protected; X=Purged;

*****
*****
*****
```

Figure 6-3 (cont). Link Map Formats



EXMPL

1986/09/09 1354:43.7

Page 9

LINK SUMMARY

All values are in hex

Load Unit Description:

| Name   | Number | Attributes | Base  | Start | Size  | Access |
|--------|--------|------------|-------|-------|-------|--------|
| DATA   |        | DF         | E0000 | 00000 | 00352 | 000    |
| EXMPL  |        | RIU        | 20000 | 2001E | 00042 | 000    |
| OVLAY1 | 0      | O          | 20042 | 20048 | 00039 | 000    |
| OVLAY2 | 1      | O          | 2002E | 20034 | 0002D | 000    |

Key to Attributes:

R=Root; D=Data; O=Fixed overlay; F=Floating overlay  
 U=Contains unresolved references; I=Contains IMAs

Bound Unit Description:

Linked for BMMU  
 Size of fixed area: 7B  
 Number of overlays; Fixed: 2, Floating: 1, Total: 3  
 Number of EDEFs: 5  
 Uninitialized data area; Size: 1F4, Initialization value: 00000000  
 Bound unit record; size: 100, count: 4

\* \* \* \* \* Bound unit contains unresolved references.

LINK DONE

\* \* \* \* \* Number of errors during the link: 1.

Figure 6-3 (cont). Link Map Formats

# NOTCMD

## NOTCMD (or ND)

Indicates that the bound unit cannot be executed as a command. This directive sets an indicator in the bound unit header area.

### FORMAT:

```
{ NOTCMD }  
{ ND }
```

ONECPU (or OU)

Indicates that the bound unit and all its subordinate tasks must be run on a single central processor unit. This directive sets an indicator in the bound unit header area.

FORMAT:

{ ONECPU }  
{ OU }

# OVERLAYTABLE

## OVERLAYTABLE (or OE or OT)

Include the name of each overlay and its associated Linker-generated overlay number in the Linker's permanent symbol table.

### FORMAT:

```
{ OVERLAYTABLE }  
{ OE           }  
{ OT           }
```

OVLY (or OY)

Assigns a specified name to the nonfloatable overlay that immediately follows, and designates the end of the preceding root or overlay. OVLY must be specified as the first directive of each nonfloatable overlay. The Linker assigns a number to each overlay. They are numbered sequentially, in ascending order; the first overlay is 0.

## FORMAT:

```
{ OVLY } name
{ OY   }
```

## ARGUMENT:

name

Name of the nonfloatable overlay that immediately follows. The overlay name can include any character that is legitimate for a file name; see the LINK directive.

## Example:

LINKER BU -PT

Load the Linker and designate BU as the bound unit name.

LINKER-rrrr-mm/dd/hhmm

Linker identification message.

L?

LINK A, B; MAP

L?

OVLY A2

Declare the end of the root (which comprises object units A.0 and B.0) and specify that the next overlay is a nonfloatable overlay named A2. The Linker assigns the number 0 to this overlay.

L?

LINK X

L?

LINK Y

L?

MAP

OVLY

L?  
QUIT

LINK DONE  
RDY:

# PAGEPOOL

## PAGEPOOL (or PL)

Indicates that the bound unit must be run in a page pool in memory. This directive sets an indicator in the bound unit header area. Before using this directive, consult with the person responsible for system building and determine available page pool memory.

This directive can be used only for bound units being linked for a VMMU environment.

### FORMAT:

```
{ PAGEPOOL }  
{ PL }
```

# PROTECT

## PROTECT (PROT or PT)

Prevents certain symbols and/or object unit names from being removed from the symbol table. Symbols that identify addresses within the range of addresses specified by the first operand through the second operand are protected. Object unit names equated to addresses within that range are protected. If a second operand is not specified, the symbol at the address of the first operand and any other symbols or object unit names equated to that address are protected. The PROTECT directive cannot be embedded in Assembly language control statements.

### FORMAT:

|                    |   |    |   |            |                   |      |   |   |   |    |   |            |                   |      |   |   |
|--------------------|---|----|---|------------|-------------------|------|---|---|---|----|---|------------|-------------------|------|---|---|
| { PROT }<br>{ PT } | } | \$ | % | X'address' | =object-unit-name | xdef | # | } | , | \$ | % | X'address' | =object-unit-name | xdef | # | } |
|                    |   |    |   |            |                   |      |   |   |   |    |   |            |                   |      |   |   |

### ARGUMENTS:

\$

Next location after the highest address of the linked root or just previously linked nonfloatable overlay.

%

Highest address+1 ever used in the linked root or any previously linked nonfloatable overlay.

X'address'

A one- to five-character hexadecimal address enclosed in apostrophes and preceded by X. The specified address is relative to either the base of the fixed area (where root and fixed overlays are linked) or to the base of the current floatable overlay. For example, if root is based at a segment address of 50000, PROTECT X'1000' indicates a segment address of 51000. Therefore, do not include a segment number as part of 'address'.

=object-unit-name

Specified object unit's base address.



xdef

Any previously defined external symbol.

#

The current address.

Example 1:

```
PROT X'1234',X'4565'
```

If the Linker is currently processing the fixed area of a bound unit based at a segment address of 70000, this directive protects all symbols and object unit names that identify addresses from 71234 through 74565.

Example 2:

```
PT =FIRST
```

This directive protects symbols that identify the base address of the object unit FIRST and all symbols equated to that address. The base address of FIRST is determined by producing a link map.

Example 3:

```
PROT SYM,X'5555'
```

If the Linker is currently processing the fixed area of a bound unit based at a segment address of 8C0000, this directive protects all symbols and object unit names that identify addresses from the address of the previously defined external symbol named SYM through 8C5555.

# PSU

## PSU

The PSU (planned segment utilization) directive can be used to indicate that each floatable overlay is to be loaded into group work space (GWS) rather than being assigned the segment address established for it by the Linker.

### FORMAT:

PSU

# PURGE

## PURGE (or PE)

Removes the following items from the symbol table:  
unprotected symbols that define addresses greater than or equal to the first address and less than or equal to the second address. If a second operand is not specified, the symbol at the address of the first operand and any other symbols or object unit names equated to that address are purged.

Limitations are:

- Undefined symbols cannot be purged.
- Symbols and object unit names that are protected by a PROTECT directive cannot be purged unless they have been unprotected (see "UNPROTECT").
- Only symbol addresses (not values) can be purged by this directive. (See "VPURGE.")
- The PURGE directive cannot be embedded in Assembly language control statements.

FORMAT:

$$\left\{ \begin{array}{l} \text{PURGE} \\ \text{PE} \end{array} \right\} \left\{ \begin{array}{l} \$ \\ \% \\ \text{X'address'} \\ \text{=object-unit-name} \\ \text{xdef} \\ \# \end{array} \right\} \left[ \left\{ \begin{array}{l} \$ \\ \% \\ \text{X'address'} \\ \text{=object-unit-name} \\ \text{xdef} \\ \# \end{array} \right\} \right]$$

ARGUMENTS:

\$

Next location after the highest address of the linked root or just previously linked nonfloatable overlay.

%

Highest address+1 ever used in the linked root or any previously linked nonfloatable overlay.

## PURGE

X'address'

A one- to five-character hexadecimal address enclosed in apostrophes and preceded by X. The specified address is relative to either the base of the fixed area (where root and fixed overlays are linked) or to the base of the current floatable overlay. For example, if root is based at a segment address of 50000, PURGE X'1000' indicates a segment address of 51000. Therefore, do not include a segment number as part of 'address'.

=object-unit-name

Specified object unit's base address.

xdef

Any previously defined external symbol.

#

The current address.

Example 1:

```
PURGE X'1234',X'4565'
```

If the Linker is currently processing the fixed area of a bound unit based at segment address 60000, this directive purges all unprotected symbol and object unit names that identify addresses from 61234 through 64565.

Example 2:

```
PE =FIRST
```

This directive purges unprotected symbols that identify the base address of the object unit FIRST and any other unprotected symbol names equated to that address.

Example 3:

```
PURGE SYM,X'5555'
```

If the Linker is currently processing the fixed area of a bound unit based at a segment address of 40000, this directive purges all unprotected symbols and object unit names that identify addresses from the address of the previously defined external symbol SYM through 45555.

# QUIT

## QUIT (or QT or Q)

Indicates that the last Linker directive has been entered. QUIT should be entered after the last overlay, or at the end of the root if there are no overlays.

If object units were successfully linked, the bound unit is completed and the Linker terminates; otherwise, the Linker terminates execution immediately.

The QUIT directive is required; it cannot be embedded in Assembly language control statements.

### FORMAT:

{ QUIT }  
{ QT }  
{ Q }

# REPORT

## REPORT (or RT)

Reports the execution environment for which the bound unit is being linked. This report includes the following information:

- The type of memory management unit under which the bound unit is intended to run.
- The base segment number used for the bound unit's root.
- The access assigned to all load units in the bound unit's fixed area (i.e., to the root and all nonfloatable overlays).
- The base segment number used for the bound unit's separated data area (if the bound unit is linked with the -R option). If you use the REPORT directive before the first object module of the bound unit is linked and you have not explicitly assigned a base segment number for the separate data portion of the bound unit, the Linker may choose to use a base segment number different from the one reported.
- The access assigned to the bound unit's separated data area (if the bound unit is linked with the -R option).

Use the REPORT directive in either of two cases:

- Before the first object module of the bound unit is linked. This allows an interactive user to decide whether to use a SEG directive to change the current segment number(s) and access before linking begins.
- After a FLOVLY directive and before the first object module of that floatable overlay is linked. In this case, the second item above is now the default base segment number the Linker will assign to this floatable overlay unless you specify a different segment number in an FSEG directive. Likewise, the third item above is the access to be assigned to this floatable overlay unless you specify different access in an FSEG directive.

FORMAT:

```
{ REPORT }  
{ RT }
```

## RERUN RELOCATABLE

### RERUN RELOCATABLE (RR)

If a shareable bound unit has to be restarted, it can be reloaded into locations other than those it occupied when the checkpoint was taken. (See the Commands manual for details on checkpoint- restart.) If this directive is not specified, the bound unit must be reloaded at the same memory pool locations it occupied when the checkpoint was taken.

If the RR directive is used, it is important to remember that after reloading, the current values of the IMAs referencing locations in the bound unit are no longer valid; therefore, if the bound unit contains IMAs (see the link map or compiler list file to determine this), RR should not be used.

#### FORMAT:

RR

# RETURN

## RETURN (or RN)

Return to accepting directives from the user-in file. This directive should be specified only in an INCLUDE file. A RETURN directive in a file specified in an INCLUDE directive is logically equivalent to an EOF mark; it returns the Linker to the user-in file.

### FORMAT:

{ RETURN }  
{ RN }



SEG (or SG)

Define the execution environment in which the bound unit will run. Before using this directive, consult with the person responsible for system building and determine the segment numbers available to task groups. If used, the SEG directive must precede any LINK/LINKN/LINKO, BASE, LDEF, PROTECT, PURGE, or UNPROTECT directives. With this directive, you can specify the segment number(s) to be assigned to the bound unit, as well as the access rights to the segment(s).

It is generally best to let the Linker assign the bound unit segment numbers. After determining which type of memory management unit the bound unit will use, the Linker assigns an appropriate base segment number to the code portion of the bound unit. If the code portion requires more than one segment, the Linker assigns successive segment numbers. After the fixed area (root and fixed overlays) has been linked, the Linker assigns the next higher segment number to the separate data portion unless a base segment number has been specified for the separated data. If the bound unit has any floatable overlays, the Linker then assigns the next higher segment number to the first floatable overlay. If the bound unit has both separated data and floatable overlays, the Linker assigns the separate data portion the segment number one higher than the highest number used for the fixed area, and then assigns the first floatable overlay a segment number one higher than the number assigned to the separate data portion.

## FORMAT:

```
{ SEG } arg1,arg2,arg3,arg4,arg5,arg6
{ SG }
```

## ARGUMENTS:

Each argument is optional, but at least one must be specified. If an argument is to be omitted and another one will be specified to the right, include a comma for the omitted argument. (e.g., SEG ,,X'A' means change only arg4.)

arg1

Specifies the type of memory management unit to be used in the execution environment. The value must be either BMMU, EMMU, or VMMU.

## SEG

### arg2

Specifies the base segment number to be used for the code portion of the bound unit. The value must be a hexadecimal number from 1 to F for a BMMU environment, from 8C to FF for an EMMU environment, and from 3B to 3FF for a VMMU environment.

Segment numbers must be specified in hexadecimal notation; e.g., X'h...'. .

### arg3

Specifies the access rights for the code portion of the bound unit. The value must be a bit string of exactly 6 binary digits for a BMMU or EMMU environment and from 6 to 10 binary digits for a VMMU environment. The value must be specified in binary string notation; e.g., B'b...'. The bit string represents the corresponding access fields in the segment descriptor. For more information on setting access, refer to "Setting Access in the Linker's SEG or FSEG Directives" earlier in this section.

### arg4

Specifies the base segment number to be used for the data portion of the bound unit, if the bound unit consists of separated code and data. The value must be a hexadecimal number from 1 to F for a BMMU environment, from 8C to FF for an EMMU environment, and from 3B to 3FF for a VMMU environment. Segment numbers must be specified in hexadecimal notation; e.g., X'h...'. (This segment number must not be the same number specified in arg2.)

### arg5

Specifies the access rights for the data portion of the bound unit, if the bound unit consists of separated code and data. The value must be a bit string of 6 binary digits for a BMMU or EMMU environment. For a VMMU environment it must be 6 to 10 binary digits.. Access rights must be specified in binary string notation; e.g., B'b...'. .

### arg6

Specifies the highest segment number in the execution environment.

## Example 1:

```
SEG BMMU,X'D',,X'F'
```

In this example, segment D with default access is assigned to the code portion of the bound unit and segment F with default access is assigned to the data portion.

## Example 2:

```
SEG EMMU,X'8E',B'001100'
```

In this example, segment 8E with ring 0 write access is assigned to the code portion of the bound unit. If the bound unit contains separated data, the default values for segment number and access apply to the separated data.

## Example 3:

```
SEG VMMU, X'3E',B'001111',X'40'
```

In this example, Segment 3E with Ring 0 write access is assigned to the code portion of the bound unit and segment 40 with default access is assigned to the data portion.

# SHARE

## SHARE (or SE)

Designates a bound unit as shareable within a memory pool. If another task requests that the bound unit be loaded, instead of another copy of the root being loaded, the existing copy in memory is used. The bound unit must have reentrant code, but the system does not check to see that it does.

SHARE causes the root of the bound unit to be assigned ring 0 write access. If the shareable bound unit contains any floatable overlays or a separate data area, the Linker leaves them with the default write access (ring 3) or with any user-supplied write access.

### FORMAT:

```
{ SHARE }  
{ SE }
```

# STACK

## STACK (or SK)

Specifies the size of the stack (as a decimal number of words). If no STACK directive is specified, the Linker uses the largest stack size specified in an object unit linked into the bound unit.

### FORMAT:

{ STACK } value  
{ SK }

### ARGUMENT:

value

The size of the stack (as a decimal number of words).

# START

## START (or ST)

Specifies the relative location within a root or overlay at which execution begins once it is loaded into memory.

If a linked object unit contains a start address (specified in an Assembler or compiler END statement), and a START directive is specified for the root or overlay containing this object unit, the Linker uses the first start address it encounters (in either a START directive or an END statement) for this root or overlay.

Once a START directive is specified, it prevents the Linker from using a start address specified in any object unit subsequently linked in the affected root or overlay. This is true even if the START directive symbol is never defined.

### FORMAT:

{ START }  
{ ST } symbol

### ARGUMENT:

symbol

Name of the externally defined symbol whose address indicates the relative address at which the root or overlay begins executing.

Default: Start address specified in the first linked object unit that has a start address. If a start address is not specified, the start address is the first non-common location in the root or overlay.

# SWAPPOOL

## SWAPPOOL (or SL)

Indicates that the bound unit must be run in a swap pool or page pool in memory. This directive sets an indicator in the bound unit header area. Before using this directive, consult with the person responsible for system building and determine available swap pool or page pool memory.

### FORMAT:

```
{ SWAPPOOL }  
{ SL }
```

# SYS

## SYS (or SS)

Indicates that the bound unit can be run as a system task. This directive sets an indicator in the bound unit header area. Before using this directive, consult the person responsible for system building and determine available system memory.

The SYS directive must be used for a bound unit that will be named in an LDBU directive to the Configuration Load Manager.

### FORMAT:

{ SYS }  
{ SS }



# UNPROTECT

## UNPROTECT (or UNPROT or UT)

Removes protection from one or more protected symbols in the symbol table. The symbols were formerly protected by means of the PROTECT directive, which prevents certain symbols and/or object unit names from being removed from the symbol table. With the UNPROTECT directive, symbols that identify addresses within the range of addresses specified by the first operand through the second operand are unprotected. Object unit names equated to addresses within that range are also unprotected. If a second operand is not specified, the symbol at the address of the first operand and any other symbols or object unit names equated to that address are unprotected. Once a symbol or object unit name is unprotected, it is eligible for subsequent purging. The UNPROTECT directive cannot be embedded in Assembly language control statements.

### FORMAT:

$$\left\{ \begin{array}{l} \text{UNPROTECT} \\ \text{UNPROT} \\ \text{UT} \end{array} \right\} \left\{ \begin{array}{l} \$ \\ \% \\ \text{X'address'} \\ \text{=object-unit-name} \\ \text{xdef} \\ \# \end{array} \right\} \left[ \begin{array}{l} \$ \\ \% \\ \text{X'address'} \\ \text{=object-unit-name} \\ \text{xdef} \\ \# \end{array} \right]$$

### ARGUMENTS:

\$

Next location after the highest address of the linked root or just previously linked nonfloatable overlay.

%

Highest address+1 ever used in the linked root or any previously linked nonfloatable overlay.

X'address'

A one- to five-character hexadecimal address enclosed in apostrophes and preceded by X. The specified address is relative to either the base of the fixed area (where root and fixed overlays are linked) or to the base of the current floatable overlay. For example, if root is based at a segment address of 50000, UNPROTECT X'1000' indicates a segment address of 51000. Therefore, do not include a segment number as part of 'address'.

## UNPROTECT

=object-unit-name

Specified object unit's base address.

xdef

Any previously defined external symbol.

#

The current address.

Example 1:

```
UNPROTECT X'1234',X'4565'
```

If the Linker is currently processing the fixed area of a bound unit based at a segment address of 8C0000, this directive unprotects all symbols and object unit names that identify addresses from 8C1234 through 8C4565.

Example 2:

```
UNPROTECT =FIRST
```

This directive unprotects symbols that identify the base address of the object unit FIRST and all symbols equated to that address. The base address of FIRST is determined by producing a link map.

Example 3:

```
UNPROTECT SYM,X'5555'
```

If the Linker is currently processing the fixed area of a bound unit based at a segment address of 70000, this directive unprotects all symbols and object unit names that identify addresses from the address of the previously defined external symbol named SYM through 75555.

## USERPOOL

### USERPOOL (or UL)

Indicates that the bound unit must be run in a user memory pool as a user task. This directive sets an indicator in the bound unit header area.

#### FORMAT:

```
{ USERPOOL }  
{ UL }
```

# VAL

## VAL (or VL)

Defines at link time a value that is equivalent to the difference between two external location definitions. When VAL is specified, the external location definitions must already exist in the Linker's symbol table.

### FORMAT:

{ VAL } symbol, external location<sub>1</sub> - external location<sub>2</sub>  
{ VL }

### ARGUMENTS:

symbol

Assign a name to the value of the distance between two locations.

external location

Externally defined location.

VDEF (or VF)

Assigns a value to an external symbol. The VDEF directive cannot be embedded in Assembly language control statements. A symbol should be defined only once, as a value or as a location. When a symbol is defined, its definition is put into the Linker symbol table so that it can be used during linking to resolve external references.

**FORMAT:**

```
{ VDEF }  
{ VF   } symbol,X'value'
```

**ARGUMENTS:**

symbol

One to eight hexadecimal characters.

X'value'

Value of the designated symbol; must be enclosed in apostrophes and preceded by X.

# VPURGE

## VPURGE (or VE)

Remove the specified external value definition from the Linker symbol table. This directive cannot be embedded in Assembly language control statements.

### FORMAT:

```
{ VPURGE } value-definition-symbol  
{ VE     }
```

### ARGUMENT:

value-definition-symbol

External symbol name associated with a particular value.

## LINKER PROCEDURES

The Linker is a system software program that functions as the final stage of program development before program execution is possible. Before being linked, each program must be compiled (or assembled) to produce an object unit (or compile unit) that the Linker identifies for linking. The Linker recognizes object units by the .O suffix (appended to each file name by the compiler). The Linker combines one or more object units to produce a bound unit. A bound unit is an executable program consisting of a root and zero or more overlays that can be loaded into memory.

This subsection describes frequently used Linker procedures. The examples provided show different methods for linking COBOL programs, including one example that uses overlays.

### Using Overlays

In situations where memory is limited, it may be necessary for you to divide your program into one or more overlays so that individual portions of your program can be called into a single memory area only when needed. Unlike the root, which cannot be reloaded once it is read into memory, an overlay can be read in as often as it is needed. See Example 4 for a link session that uses overlays.

### Interrupting Linker Execution

If at any time during Linker execution you want to interrupt processing, you can perform one of the following actions:

- Press the QUIT, INTERRUPT, or BREAK key at your terminal.
- Enter C Bid if you are at the operator terminal, where id is your two-character task group identification.

After you perform one of the above actions, a **\*\*BREAK\*\*** message appears on your terminal. You can now:

- Enter any valid ECL command.
- Resume Linker execution as if no break had occurred by entering the Start (SR) command.
- Terminate Linker processing and return to command level by entering the Unwind (UW) command.
- Restart your task group by issuing a New Process (NEW\_PROC) command.

If you want to terminate the MAP operation and jump to the next Linker directive, issue a Program Interrupt (PI) command.

## Sample Link Sessions

The sample link sessions that follow will help you become familiar with Linker procedures.

The first three examples illustrate different methods for linking a COBOL program. The fourth example describes a method for linking a COBOL program that contains two overlays.

### Example 1: Linking With a Minimum of Directives

This example illustrates a link session requiring a minimum of Linker directives.

The COBOL program DATIME has just been compiled. A List (LS) command is issued to examine the contents of the programmer's (Cook) working directory:

LS

Directory: ^COBOL>ONE  
Time: 1986/01/17 1103:33

|          |     |    |
|----------|-----|----|
| DATIME.C | SEQ | 4  |
| DATIME.L | SEQ | 10 |
| DATIME.O | SEQ | 6  |

Total Sectors: 20

The file DATIME.C contains Cook's source program. The files DATIME.O and DATIME.L were produced by the COBOL compiler. DATIME.O contains the object unit that the Linker uses to produce a bound unit named DATIME.

Cook now wants to link his program into a bound unit. He enters the command:

```
LINKER DATIME -PT -COUT !LPT00 -R
```

Cook has specified that he wants to create a bound unit named DATIME. Cook also specifies that his link map be directed to printer !LPT00 rather than to the file named DATIME.M in his working directory. (The contents of DATIME.M are described later in this example.) The -PT argument causes the Linker prompt L? to appear when the Linker is ready to receive input. It is recommended that new users include this argument in the Linker command format. The -R argument indicates that all common blocks are to be placed into a separate data area.

The Linker responds:

LINKER-(Linker identification)



Cook now enters Linker directives. Each directive has been keyed to the explanatory notes that follow. (The Linker prompts have been omitted from the text.)

```
LSR
*****
PRIM WORKING DIRECTORY
*****
LIB ^DEVX>NEW>ZCART
LSR
*****
PRIM WORKING DIRECTORY
LIB ^DEVX>NEW>ZCART
*****
LINK DATIME
MAPD
QUIT
LINK DONE
```

Cook asks the Linker to list the search rules it currently uses to locate object units to be linked. The Linker's response indicates that the primary directory searched is Cook's working directory.

The LSR directive is optional. Omitting it does not affect the linking process. It is included here and below to illustrate how you can view the Linker's current search rules as they result from your use of LIB directives.

Because Cook's COBOL program requires the run-time routines located in the directory ZCART, Cook must designate ZCART as the secondary directory to be searched by the Linker. If the required object units cannot be found in Cook's primary directory, the Linker automatically searches the secondary directory ^DEVX>NEW>ZCART.

Cook lists the Linker's modified search rules.

This LINK directive queues the object unit DATIME.O for linking.

The MAPD directive produces a link map that is written out to the printer LPT00 (as specified in the -COUT argument of the Linker command). This link map is shown in Figure 6-4. It also causes DATIME.O to be linked before the map is produced.

Cook enters the QUIT directive to indicate that there are no more directives. The Linker builds the bound unit and terminates.

LINKER-(Linker identification) (System identification)  
 Bu= DATIME Linked on: 1986/01/15 1826:47.6 -R

-> LIB DEVX>NEW>ZCART

-> LINK DATIME

-> MAPD

DATIME.O  
 DATIM (00020000)  
 13COBOLM REV. 1.0 DATE 86/01/13 TIME 1830

[ LINK04 ZCRMPI]

[ EDEF DATIME]

[ LINK04 ZCRMDT]

[ LINK04 ZCRMIN]

[ LINK04 ZCRMDI]

DEVX>NEW>ZCART>ZCRMPI.O  
 ZCRMPI 8510100 (000200BE)  
 HRS ASSEMBLER 10.1 10/10/85 1651.2 edt Thu

[ LINK ZCRMER ]

[ LINK ZCRASP ]

DEVX>NEW>ZCART>ZCRMDT.O  
 ZCRMDT 8508240 (00020469)  
 HRS ASSEMBLER 10.1 08/24/85 1057.3 edt Sat

DEVX>NEW>ZCART>ZCRMIN.O  
 ZCRMIN 8508240 (000204DB)  
 HRS ASSEMBLER 10.1 08/24/85 1100.2 edt Sat

DEVX>NEW>ZCART>ZCRMDI.O  
 ZCRMDI 8601130 (00020803)  
 HRS ASSEMBLER 10.1 01/13/86 1559.9 est Mon

DEVX>NEW>ZCART>ZCRMER.O  
 ZCRMER 8511130 (00020907)  
 HRS ASSEMBLER 10.1 11/13/85 1414.3 est Wed

[ LINK ZCPROT ]

DEVX>NEW>ZCART>ZCRASP.O  
 ZCRASP 8601130 (00020992)  
 HRS ASSEMBLER 10.1 01/13/86 1439.6 est Mon

DEVX>NEW>ZCART>ZCPROT.O  
 ZCPROT 8511210 (000209AA)  
 HRS ASSEMBLER 10.1 11/21/85 1436.5 est Thu

-> QT

Figure 6-4. Sample Link Map (DATIME.M)

\*\*\*\*\*  
\*\*\*\*\* M A P \*\*\*\*\*  
\*\*\*\*\*

\*\*Start: 00020016  
\*\*Low: 00020000  
\*\*High: 000209CC  
\*\*\$LCOMW: 000E0000  
\*\*Current: 000209CC

\*\*\*\*\* Common Block Definitions \*\*\*\*\*

\*\* DATIME 00020000  
\* DATIME 00020000  
L \$LCOMW 000E0000 L LUDATA 000E00BB L LGDATA 000E029C

\*\*\*\*\* External Definitions \*\*\*\*\*

P ZHCOMM 00000000 P ZHREL 00020000  
\*\* DATIME 00020000  
\* DATIME 00020000  
DATIME 00020016  
\* ZCRMPI 000200BE  
ZCRINI 000200BE ZCRMEX 000203D8 ZCRMSG 00020409  
ZCSTOP 00020293 ZCGRNG 0002035C ZCXRNG 00020366  
ZCB32K 00020310 ZCERR0 000203C3 ZCERRX 000203C7  
\* ZCRMDT 00020469  
ZCRMDT 00020476  
\* ZCRMIN 000204DB  
ZCRMIN 000204DB  
\* ZCRMDI 00020803  
ZCRMY1 00020805 ZCRMY0 000208EC ZCRMY2 000208EE  
ZCRMY3 000208F6  
\* ZCRMER 00020907  
ZCRMER 0002091D  
\* ZCRASP 00020992  
ZCRASP 00020992  
\* ZCPROT 000209AA

Key: \*\*=Root or overlay name, or heading; \*=Object file name; C=Common;  
L=Local common; D=Displacement reference; V=Value; P=Protected; X=Purged;

\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*

Figure 6-4 (cont). Sample Link Map (DATIME.M)

LINK SUMMARY

All values are in hex

Load Unit Description:

| Name   | Number | Attributes | Base  | Start | Size  | Access |
|--------|--------|------------|-------|-------|-------|--------|
| DATA   |        | DFI        | E0000 | 00000 | 0029C | 000    |
| DATIME |        | RI         | 20000 | 20016 | 009CC | 000    |

Key to Attributes:

R=Root; D=Data; O=Fixed overlay; F=Floating overlay  
 U=Contains unresolved references; I=Contains IMAs

Bound Unit Description:

Linked for BMMU  
 Size of fixed area: 9CC  
 Number of overlays; Fixed: 0, Floating: 1, Total: 1  
 Number of EDEPs: 1  
 Uninitialized data area; Size: 1E1, Initialization value: 00000000  
 Bound unit record; size: 100, count: D

LINK DONE

Figure 6-4 (cont). Sample Link Map (DATIME.M)

The linking process has been successfully completed; Cook now enters an LS command to examine the contents of his working directory:

LS

Directory: ^COBOL>ONE  
 Time: 1986/01/17 1119:38

|          |     |    |
|----------|-----|----|
| DATIME   | F R | 28 |
| DATIME.C | SEQ | 4  |
| DATIME.L | SEQ | 10 |
| DATIME.O | SEQ | 6  |

Total Sectors: 48

The bound unit DATIME now resides in Cook's working directory, ready for execution. Note that DATIME.M, the link map, is not listed in the working directory. This file was written out to the printer LPT00 when cook issued the MAPD directive.

## Example 2: Specifying an Input Device

This example shows you how to specify a directive input device (such as a file, another terminal, or card reader) from which the Linker reads its directives. This procedure is useful if you have many directives to enter, or if you wish to create a Linker directive file for future use.

Programmer Cook wants to have the Linker read its directives from a file named LKDIR. He invokes the Editor, types in his Linker directives, and writes the file to the pathname ^DEVX>WORK>LKDIR. The contents of LKDIR are listed below. (The object unit to be linked, DATIME.O, resides in Cook's working directory.)

```
LIB ^DEVX>NEW>ZCART
LINK DATIME
MAP
QUIT
```

To activate Linker processing, Cook need only enter the following command:

```
LINKER DATIME -IN ^DEVX>WORK>LKDIR
```

Cook has specified that he wants to create a bound unit named DATIME. The -IN argument specifies the pathname of the file from which Linker directives are read. Cook could also have designated another terminal or a card reader as the directive input device.

The complete dialog as it appears at Cook's terminal is shown below:

```
LINKER DATIME -IN ^DEVX>WORK>LKDIR
LINKER-(Linker identification)
LINK DONE
RDY:
```

## Example 3: Linking More Than One Object Unit

In this example, Cook wants to link the object unit DATIME.O, which resides on a diskette volume named ^DSK, and whose full pathname is ^DSK>MYDIR>DATIME.O. Since Cook wants to create the bound unit DATIME in his working directory ^SYSRES>WORK, he must designate the directory ^DSK>MYDIR as the primary directory the Linker will search.

A second object unit NEXT.O is also to be linked into the bound unit. It resides on the current working directory. Cook wants to link NEXT.O to DATIME.O after DATIME.O has been linked.

Cook's dialog with the system is shown below. The dialog has been keyed to the explanatory notes that follow.

LINKER DATIME

LINKER-(Linker identification)

LIB ^DEVX>NEW>ZCART

IN ^DSK>MYDIR

LSR

\*\*\*\*\*

PRIM ^DSK>MYDIR

LIB ^DEVX>NEW>ZCART

\*\*\*\*\*

LINK DATIME

IN

LSR

\*\*\*\*\*

PRIM WORKING DIRECTORY

LIB ^DEVX>NEW>ZCART

\*\*\*\*\*

LINK NEXT

MAP

QUIT

LINK DONE

LS

Directory: ^SYSRES>WORK

Time: 1986/01/17 1148:33

|          |     |    |
|----------|-----|----|
| DATIME   | F R | 28 |
| DATIME.M | SEQ | 18 |
| NEXT.O   | SEQ | 6  |

Total Sectors: 52

Invoke the Linker and specify DATIME as the name of the bound unit to be created.

Request that the Linker search the secondary directory ZCART for the required COBOL run-time routines.

Specify the IN directive, designating ^DSK>MYDIR as ^DSK>MYDIR the primary directory in which the Linker should search for the required object unit.

List the Linker's search rules. The Linker's response indicates that the primary directory to be searched is ^DSK>MYDIR.

Request that DATIME.O be queued for linking. Enter the IN directive again, this time omitting a pathname. This action modifies the Linker's search rules; the primary directory to be searched is now Cook's working directory.

List the Linker's search rules again, and note that the Linker's primary directory has been redirected to his working directory.

Request that NEXT.O be queued for linking. Issue a MAP directive. A link map is written to a file named DATIME.M in Cook's working directory.

Indicate that there are no more Linker directives.

Specify an LS command to verify that the bound unit DATIME has been created in his working directory. DATIME.M contains link map information.

#### Example 4: Linking with Two Overlays

This example describes how to link a program containing two overlays.

Programmer Shepard has written a COBOL program called PROGR, which calls two overlays, PROG0 and PROGL. Figure 6-5 shows the relationship between the root PROGR and the two overlays. Source listings of the root program and the two overlays are shown in Figures 6-6, 6-7, and 6-8. (Source listings are included to show you the relationships that exist between a root program and its overlays. In Figure 6-6, for example, note how the source program calls in its overlays. If Shepard's link is successful, each greeting message will join with the others in the specific order Shepard intends.)

Following COBOL compilation of all three source files, Shepard issues an LS command to display the contents of the working directory:

LS

Directory: ^COBOL>TWO  
Time: 1986/01/17 1200:08

|         |     |    |
|---------|-----|----|
| PROG0.C | SEQ | 4  |
| PROG0.L | SEQ | 8  |
| PROG0.O | SEQ | 6  |
| PROGL.C | SEQ | 4  |
| PROGL.L | SEQ | 8  |
| PROGL.O | SEQ | 6  |
| PROGR.C | SEQ | 4  |
| PROGR.L | SEQ | 10 |
| PROGR.O | SEQ | 10 |

Total Sectors: 60

Note that all three object units to be linked are located in Shepard's working directory. PROGR.O is the object unit that forms the root of the bound unit SAMPLE. PROG0.O and PROGL.O are object units that form the two overlays of the bound unit. Figure 6-5 shows the bound unit Shepard will create when she links the root module PROGR.O and the two overlay modules, PROG0.O and PROGL.O.

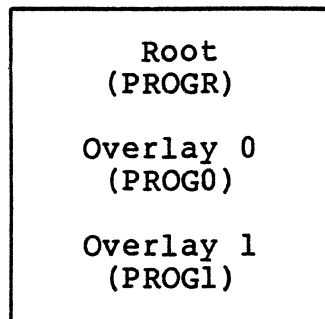


Figure 6-5. Structure of the Bound Unit SAMPLE

```

IDENTIFICATION DIVISION.
PROGRAM-ID. PROGR.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. DPS6.
OBJECT-COMPUTER. DPS6.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ME PIC X(17) VALUE "PROGR IN THE ROOT".
PROCEDURE DIVISION.
BEGIN.
    DISPLAY "THIS IS " ME.
    DISPLAY "PROGR ATTEMPTING TO CALL PROG0 IN OVERLAY 0".
    CALL "PROG0" USING ME.
    DISPLAY "THIS IS " ME.
    CANCEL "PROG0".
    DISPLAY "PROGR ATTEMPTING TO CALL PROG1 IN OVERLAY 1".
    CALL "PROG1" USING ME.
    DISPLAY "THIS IS " ME.
    CANCEL "PROG1".
    DISPLAY "DONE".
    STOP RUN.
  
```

Figure 6-6. Source Listing of PROG



```

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG0.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. DPS6.
OBJECT-COMPUTER. DPS6.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ME PIC X(18) VALUE "PROG0 IN OVERLAY 0".
LINKAGE SECTION.
01 CALLER PIC X(17).
PROCEDURE DIVISION USING CALLER.
BEGIN.
    DISPLAY "THIS IS " ME.
    DISPLAY "    CALLED BY " CALLER.
EXIT PROGRAM.

```

Figure 6-7. Source Listing of First Overlay Module PROG0

```

IDENTIFICATION DIVISION.
PROGRAM-ID. PROG1.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. DPS6.
OBJECT-COMPUTER. DPS6.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ME PIC X(18) VALUE "PROG1 IN OVERLAY 1".
LINKAGE SECTION.
01 CALLER PIC X(17).
PROCEDURE DIVISION USING CALLER.
BEGIN.
    DISPLAY "THIS IS " ME.
    DISPLAY "    CALLED BY " CALLER.
EXIT PROGRAM.

```

Figure 6-8. Source Listing of Second Overlay Module PROG1

Shepard is ready to link her program. Her dialog with the system is described below. The dialog has been keyed to the explanatory notes that follow.

```
LINKER SAMPLE
LINKER-(Linker identification)
LIB ^SYSVOL>LDD>ZCART
LINK PROGR
MAP
OVLY PROG0
BASE $
LINK PROG0
MAP
OVLY PROGL
BASE =PROG0
LINK PROGL
MAP
QUIT
LINK DONE
```

Invoke the Linker, specifying SAMPLE as the name of SAMPLE the bound unit to be created.

Request the Linker search the secondary directory ZCART for the required COBOL run-time routines.

Queue the object unit PROGR.O for linking.

Specify a MAP directive. A link map is written to a file named SAMPLE.M in Shepard's working directory.

Designate the end of the root and the beginning of the first overlay PROG0.

Identify the relative load address for PROG0 within the bound unit. The BASE \$ directive specifies that PROG0 will be linked beginning with the next location after the highest address of the root module PROGR. This is the default base address for PROG0. This BASE directive could be omitted.

Queue the object unit named PROG0 for linking.

Specify a MAP directive.

Designate the end of the overlay PROG0 and the beginning of the second overlay PROGL.

Request that the overlay named PROGL be loaded starting at the same relative address as the object unit PROG0. Overlays PROG0 and PROGL can never be in memory at the same time.

Queue PROGL for linking.

Request a link map.

Terminate Linker processing.

Shepard is now ready to execute the bound unit SAMPLE. (No data files are required.) Shepard enters the bound unit name:

SAMPLE

The program responds:

THIS IS PROGR IN THE ROOT.  
PROGR ATTEMPTING TO CALL PROG0 IN OVERLAY 0.  
THIS IS PROG0 IN OVERLAY 0.  
    CALLED BY PROGR IN THE ROOT.  
THIS IS PROGR IN THE ROOT.  
PROGR ATTEMPTING TO CALL PROG1 IN OVERLAY 1.  
THIS IS PROG1 IN OVERLAY 1.  
    CALLED BY PROGR IN THE ROOT.  
THIS IS PROGR IN THE ROOT.  
DONE.



REMOVE THIS PAGE AND PLACE TAB FOR

TAB 7

MULTIUSER D.B. SYSTEM



## *Section 7*

# **MULTIUSER DEBUGGER (SYMBOLIC MODE)**

The Multiuser Debugger is a general purpose tool used for testing application programs. The debugger operates in two modes:

- Numeric Mode--primarily used for applications written in Assembly language (can be used on any program)
- Symbolic Mode--used for applications written in higher-level languages such as COBOL or FORTRAN.

Numeric debugging is described in Section 8 of this manual. Symbolic debugging is described in this section; full information on symbolic debug functions and directives is provided. Procedural information on using symbolic debugging directives to debug a bound unit is also included in this section.

### OVERVIEW

The debugger can be used with object units that have been compiled with the debug option (-SYMBOL). The debug option causes the compiler to generate a file for later use by the debugger.

Additionally, the object unit must be linked with the Linker's debug option (-SYMBOL). In any bound unit there can be a mixture of programs compiled with and without the debug option.

## CAPABILITIES

The debugger uses the object unit tables and a Linker symbol table to manipulate breakpoints, process action lines, and alter and display (dump) data variables. The debugger uses the same referencing format for variables as in the source programs. This referencing format can be a variable name, label, or line number.

The various debug directives can be used to halt a program at selected breakpoints during execution, restart the program from the same point, or change sequence and start from a different point. While the program is halted, you can examine and alter program data and set further breakpoints.

- \* The debugger can be used to debug COBOL and FORTRAN programs. Programs must be compiled with the debug option (-SYMBOL). This option generates an object unit file.

- \* The debugger uses the object unit symbol tables and link symbol table (produced by the Linker debug option, -SYMBOL) to manipulate breakpoints, process action lines, and alter data variables. The symbol table is called object unit name.Z. The special link map file is called bound unit name. Object unit name is the name of the COBOL or FORTRAN source program. Do not try to edit these symbol table files because you may destroy necessary information.

### NOTE

All concurrent users of the multiuser debugger must use a common copy of the DEBUG bound unit. Attempting to use a different copy of DEBUG from the one currently being used results in a 5D22 error. For this reason, you should not make a copy of DEBUG.

## INVOKING THE DEBUGGER (SYMBOLIC MODE)

After the program to be debugged has been compiled and linked with the debug option, you can invoke the debugger with the following command:

### FORMAT:

DEBUG program\_name

### ARGUMENT:

program\_name

Name of the bound unit to be debugged.



To use the debugger effectively, you should become familiar with the directives, terms, and symbols listed in Tables 7-1 through 7-3. Table 7-1 lists the debugger directives by function, indicating the directive name and its meaning. Table 7-2 is a list of terms used in debugger directives. Table 7-3 is a list of debugger special symbols and their meanings.

Table 7-1. Summary of Symbolic Mode Directives

| Function                         | Directive Name                | Meaning   |
|----------------------------------|-------------------------------|---|
| Breakpoint control               | AT                            | Set breakpoints   |
|                                  | CLEAR                         | Clear specified or all breakpoints  |
|                                  | LIST                          | List current breakpoints  |
| Trace trap control               | TRACE                         | Trace flow of program   |
| Display and modification of data | CHANGE                        | Change specified variable's control contents  |
|                                  | DUMP                          | Display specified variable (dump)   |
|                                  | SET                           | Set values represented by special symbols   |
| General execution                | GO                            | Resume execution  |
|                                  | IF                            | Conditional requirements for breakpoint and request lists                                   |
|                                  | MODE                          | Switch between symbolic and numeric modes   |
|                                  | ACTIVATE                      | Change reference to a different object unit   |
|                                  | PAUSE                         | Enter interactive mode  |
|                                  | QT                            | Terminate debugger (quit)   |
|                                  | SP                            | Temporarily suspend the Multiuser Debugger; return control to the command processor (sleep) |
| STEP                             | Execute one program statement |   |

Table 7-2. Terms Used in Symbolic Mode Directives

| Term             | Definition  |
|------------------|---|
| Character string | A string of characters enclosed in apostrophes (') or quotes. The string may include all printable characters except those used for terminal editing. Use two apostrophes to include an apostrophe in the string. |
| Directive        | A statement to the debugger containing keywords, which cause the debugger to perform a specified action (e.g., AT, CLEAR, TRACE).   |
| Hex value        | A maximum of eight hexadecimal digits prefixed by a percent (%) sign.   |
| Identifier       | A name containing a maximum of 30 characters. Valid characters include all alphanumerics, the hyphen, and the underscore. The first character must be alphabetic.   |
|                  | <p style="text-align: center;">NOTE</p> <p>A minus sign in an expression must be preceded and followed by a blank to distinguish it from a hyphen.</p>  |
| Input unit       | A line consisting of one or more debugger directives separated by semicolons. The maximum length is the input device's maximum line length.   |
| Integer          | A value less than 65535 and greater than or equal to -65536 entered as a string of decimal digits.  |
| Statement        | A single source statement that generates executable code.   |
| Type             | Identification of the internal storage and external display formats of variables. All source-defined variables have a basic type such as integer, real, character, string, or alphabetic.                         |
| Variable         | A single field, array element, entire array, record, or single component of a record.   |

Table 7-3. Symbolic Mode Special Symbols

| Symbol    | Definition                            |
|-----------|---------------------------------------|
| \$R1-\$R7 | Index registers (containing data)     |
| \$B1-\$B7 | Base registers (containing addresses) |
| \$P       | Program counter                       |
| \$        | Current breakpoint                    |
| \$T       | True                                  |
| \$F       | False                                 |
| \$L       | Prefix for numeric statement labels   |

The following is a list of reserved keywords. These reserved keywords and special symbols should not be used as variable names or labels in programs to be run with the debugger.

|          |           |
|----------|-----------|
| AT       | P         |
| CH       | QT        |
| CHANGE   | SP        |
| C        | -NUM      |
| CLEAR    | -NUMERIC  |
| DUMP     | -SYMBOLIC |
| IF       | \$        |
| L        | SP        |
| LIST     | \$R1-\$R7 |
| MODE     | \$B1-\$B7 |
| ACTIVATE | \$L       |
| PAUSE    | \$F       |
| SET      | TR        |
| TRACE    |           |

After invoking the debugger, you are prompted with the greater-than character (>). To initialize the debugger, set, list, and clear breakpoints using the AT, LIST, and CLEAR directives.

After you have arranged breakpoints to your satisfaction, use the SP directive to return to the Command Processor. Execute your program normally. When a breakpoint is reached, program execution is suspended and the debugger enters interactive mode. Now you can enter any valid debugger directive to debug your program.

To leave the debugger, either continue execution of the program to the normal end (perhaps clearing unwanted breakpoints with the CLEAR directive) and then enter the GO directive, or enter the QUIT directive which terminates debugger control and resumes normal execution of your program.

While the debugger is in operation, it maintains two internal variables that identify the current bound unit and the current debugging mode:

- current object unit identifies the object unit to which symbolic debugging directives refer. The initial default is the first linked object module with a symbol table. During execution of a program, current object unit is automatically changed to identify the object unit in which the most recent breakpoint occurred. You can override the automatic setting with the SET directive.
- current mode is automatically set to symbolic if the debugger is invoked with a bound unit name.

#### DEBUGGER AND BREAK KEY FUNCTIONALITY

Typing DEBUG after pressing the BREAK key and getting the \*\*BREAK\*\* message transfers you to the debugger. To return to the previous level, enter the SP directive or terminate the debugger completely with the QUIT directive.

If DEBUG was the task that was broken, the responses allowed are:

- Any command
- UW, PI, SR or NEW\_PROC.

#### NOTES

1. The PI response returns the user group to the debugger input mode and allows the entry of debugger directives.
2. If the Debugger task was broken and DEBUG was entered as the response, the user would be placed in the debugger input mode.
3. The UW response causes the debugger to be exited as if a GO or SP directive was input, depending on which was appropriate for the current state of the debugger. That is, if debug was invoked as the result of encountering a breakpoint, a GO is appropriate to exit debug.

## PLANNING CONSIDERATIONS

### Controlling Execution of the User's Program

The following directives can be used to control execution of the user program.

IF is used to specify if-statement processing.

GO causes execution to resume.

STEP steps through a program one executable statement at a time.

PAUSE enters input mode and allows you to request display of variables and registers.

PAUSE used with IF causes execution to stop and entry into interactive mode.

### Setting Breakpoints

You can set, list, and clear breakpoints during initialization as well as during execution. All other functionality can only be done during execution. The AT directive sets breakpoints; CLEAR deletes them. The LIST directive lists breakpoints.

### Monitoring the Value of Variables

The IF directive can be used in a number of ways to perform other directives when a certain condition is met. The primary use of the IF directive is to monitor the value of a variable during execution of the user's program. If the value meets a given condition, program execution stops and the user is notified.

### Controlling Output

The DUMP directive displays the contents of variables and constants.

### Maintaining a Trace History

The TRACE directive controls tracing program execution. It can be used in conjunction with IF for conditional tracing. You can issue the TRACE directive only while stopped at a breakpoint.

### Altering Values

The CHANGE directive alters the values of variables. SET sets or alters the values of special symbols.

## DEBUGGER DIRECTIVES

This subsection provides an alphabetic listing of the debugger directives with detailed descriptions for each directive.

ACTIVATE (OR AC)

Change reference to a different object unit.

## FORMAT:

```
{ ACTIVATE } object_unit_name[/overlay_name]
{ AC }
```

## ARGUMENTS:

object\_unit\_name

Name of the object unit other than the one currently active.

overlay\_name

Number of the overlay in which the object unit is linked.

## DESCRIPTION:

The object unit named becomes the current object unit.

## Example:

```
AC PROC1
DUMP ABC
AC PROC2
DUMP DEF
AC PROC3/4
DUMP XYZ
```

ABC, DEF, and XYZ are variables declared in three separately compiled object units. This sequence of directives displays ABC, activates the PROC2 symbol block, and displays DEF. Then the symbol table for PROC3, linked in overlay 4, is activated, and variable XYZ is displayed.

## AT

### AT

Set breakpoints in the program.

#### FORMAT:

```
AT location_list [(request_list)]
```

#### ARGUMENTS:

##### location\_list

One or more places in the program where you want to set a breakpoint. Location specifiers are separated by commas. Individual statements in the program are identified either by statement label or source line number. Set breakpoints only on executable statements. For FORTRAN programs, do not set breakpoints on FORMAT statements.

##### request\_list

Optional list of one or more directives to be executed when a breakpoint is reached. A request list can be a single directive or a series of directives delimited by parentheses. Directives in a request list are separated by either semicolons or newline characters. GO is understood to be the last directive in a breakpoint request list. If no request list is given, PAUSE is assumed.

#### DESCRIPTION:

A breakpoint is set in the program for every statement in the location list. A breakpoint identifier is assigned to each breakpoint, and a brief message is printed showing the line number, label (if any), and assigned identifier number. Breakpoint identifiers are numbers assigned in descending order beginning with 31. You can set a maximum of 32 breakpoints (from 31 through 0, inclusive). Request list directives are saved in the DEBUG.SM file for each breakpoint. To distinguish FORTRAN statement labels from line numbers, prefix statement labels with "\$L".



## Example 1:

```
AT 1020,LOOP (PAUSE)
AT 1020,LOOP
```

These two directives are equivalent. Both cause program breakpoints to occur before execution of statement number 1020 and the statement labeled LOOP. When a breakpoint occurs, the debugger enters interactive mode and prompts you (with the greater-than (>) sign) to enter directives from the terminal.

## Example 2:

```
AT LOOP1 (DUMP INVENT_PART_NO,A(J_INDEX),J_INDEX;CH BUFNO=1)
```

Each time execution reaches LOOP1, the DUMP and CHANGE directives are performed, and execution of the target program resumes without user intervention.

## Example 3:

```
AT 2/1
```

This example illustrates how to set breakpoints within include (COPY) files in COBOL. In this case, the breakpoint is set in the first line within the second COPY file.

# CHANGE

## CHANGE (OR CH)

Alter the contents of variables.

### FORMAT:

```
{ CHANGE } change_list  
{ CH }
```

### ARGUMENT:

change\_list

List of change statements of the form:

```
variable name      { decimal integer  
record component = { hexadecimal string  
                   { character string  
                   { $T  
                   { $F
```

### DESCRIPTION:

For each change statement, the item on the left is assigned the value of the element on the right. Change statements are separated by commas.

### Example:

```
CHANGE TAGA = %3031, TAGC = 5
```

The hexadecimal value 3031 is placed in TAGA and 5 is loaded into TAGC.

The data types of the left side and right side must match. For example, a variable name defined as a hexadecimal string cannot be changed to a decimal integer. The one exception is that any type variable can change to a hexadecimal string that represents exactly the internal format of the data. There is no other conversion of data types.

# CLEAR

## CLEAR (OR C)

Delete breakpoints.

FORMAT:

$\left. \begin{array}{l} \text{CLEAR} \\ \text{C} \end{array} \right\} \text{ breakpoint\_list}$

ARGUMENT:

breakpoint\_list

List, which can include:

- Individual breakpoint numbers
- A range of breakpoints (e.g., N1 to N2)
- \$ (indicating the current breakpoint)
- \* (indicating all currently defined breakpoints).

DESCRIPTION:

All breakpoints specified in the breakpoint list are deleted.

Example 1:

CLEAR \*

Clear all currently defined breakpoints.

Example 2:

C \$, 14, 15, 0 TO 5

Clear multiple breakpoints.

# DUMP

## DUMP (OR DP)

Display program variables and other information about the program execution.

### FORMAT:

$\left. \begin{array}{l} \text{DUMP} \\ \text{DP} \end{array} \right\}$  dump\_list

### ARGUMENT:

dump\_list

List of items separated by commas, which can include:

- Variable names representing individual values, or record structures:

DUMP HEAD\_OF\_LIST, TABLE, MASS\_REC

- Range of variables as declared textually in the program:

DUMP ABC TO J\_MODE

- Record component:

DUMP C OF B OF A

DUMP C IN B IN A (these two are equivalent).

If # precedes a variable name, the variable is displayed in hexadecimal.

### DESCRIPTION:

Each requested item is displayed on a new line. Record components are indented according to level. The item name is printed on the left followed by the item value. The value is printed in a format conforming to its data type unless the hexadecimal override character (#) precedes the name.

### Example:

DUMP ABC, 'RESULT'

Display the contents of variable ABC and print the character string RESULT.

GO

GO

Resume execution of the program.

FORMAT:

GO [program\_location]

ARGUMENT:

program\_location

Either a statement label or a statement line number.

DESCRIPTION:

If no argument is given, the debugger resumes execution of the program where it left off. If a program location is given, the debugger resumes execution at the new location.

NOTE

The program location option should be used with caution because registers used by the bound unit can be scrambled by such a jump.

# IF

## IF

Provide a simple conditional for use in breakpoint and trace request lists.

### FORMAT:

IF variable  $\left\{ \begin{array}{l} = \\ \hat{=} \\ > \\ > \\ < \\ < \end{array} \right\} \left\{ \begin{array}{l} \text{character string} \\ \text{hexadecimal string} \\ \text{decimal literal} \\ \$T \\ \$F \end{array} \right\}$  request\_list

### ARGUMENTS:

= Equals

$\hat{=}$  Not equal

> Greater than

>= Greater than or equal to

< Less than

<= Less than or equal to

request\_list

Required list of one or more directives to be performed when the IF expression evaluates to True. A request list consists of a single directive or a series of directives delimited by parentheses. Directives in a request list are separated by either semicolons or new line characters.

### DESCRIPTION:

When an IF directive is performed, the comparative expression is evaluated. If the result is True, the request list is performed. When execution of the request is completed (without encountering a GO or PAUSE), processing continues with the next directive following the list.

If the result of the comparison is False, the request list is ignored.

The data types on the left and right of the relation must match. For example, a variable name defined as a hexadecimal string cannot be compared to a decimal literal. The one exception is that any type variable can be compared to a hexadecimal literal that represents exactly the internal format of the data. There is no other conversion of data types.

The length of a comparison corresponds to the language rules. If an expression being evaluated is determined to be invalid, an error message is issued and the debugger pauses.

Examples:

```
AT 1020 (IF VAR1 = 'ABCD' (PAUSE))
AT 1020 (IF VAR1 = %41424344 (P))
```

These two directives are both valid assuming the base type of VAR1 is a character string. A True condition causes a PAUSE; False causes a GO.

```
AT 1020 (IF BOOL = $F (PAUSE))
```

The debugger pauses only if BOOL is False.

```
TR (IF J_INDEX < 0 (PAUSE))
```

Establish tracing for all statements in the compiled program. At each statement, the IF expression is evaluated. If J\_INDEX is ever less than zero, the PAUSE occurs. Otherwise, the program continues.

# LIST

## LIST (OR L)

List current breakpoints.

FORMAT:

$\left. \begin{array}{l} \text{LIST} \\ \text{L} \end{array} \right\} [\text{breakpoint\_list}] \quad [-\text{LG}]$

ARGUMENT:

breakpoint\_list

List, which can include:

- Individual breakpoint numbers
- A range of breakpoints (e.g., N1 to N2)
- \* (indicating all currently defined breakpoints)
- \$ (indicating the current breakpoint).

DESCRIPTION:

The breakpoint identifier (an integer between 0 and 31), the line number, and label (if any) are printed for all specified breakpoints. In addition, if -LG is given, the request list (if any) associated with each breakpoint is printed.

Example 1:

LIST 3,4,10 TO 15 -LG

Print the basic information and request lists for breakpoints 3, 4, and 10 through 15.

Example 2:

L\$

Print the basic information for the breakpoint at which the program is currently stopped.



MODE

Define the debugger mode of operation desired: symbolic mode for debugging programs written in high-level languages, numeric mode for debugging Assembly language programs.

## FORMAT:

$$\text{MODE} \left\{ \begin{array}{l} \{ \text{NUMERIC} \} \\ \{ \text{NUM} \} \\ \{ \text{SYMBOLIC} \} \\ \{ \text{SYM} \} \end{array} \right\} .$$

## DESCRIPTION:

If the debugger is currently in symbolic mode, type MODE NUMERIC to put the debugger in numeric mode. See Section 8 of this manual for a description of numeric directives.

If the debugger is currently in numeric mode, type MODE SYMBOLIC to put the debugger in symbolic mode. Symbolic mode can only be used with bound units that have been compiled and linked with the -SYMBOL control argument. In addition, you can only change from numeric mode to symbolic mode if the debugger is initially invoked in symbolic mode.

# PAUSE

## PAUSE (OR P)

Enter interactive mode.

FORMAT:

```
{ PAUSE }  
{ P }
```

DESCRIPTION:

When a PAUSE directive is performed, the debugger enters interactive mode, sends a prompt message (the greater-than sign (>)) to the user-in file, and reads the user-out file (generally a terminal) to obtain its next directive.

When a PAUSE is encountered within a request list, it takes effect immediately, and any directives remaining in the list are ignored.

Example:

```
AT 1020 (DUMP I,NEXT;IF NEXT=%40 (PAUSE))
```

Whenever line 1020 is reached, the variables I and Next are dumped. Then, the IF expression is evaluated. If NEXT is a hexadecimal 40, the debugger pauses. Otherwise, execution of the program resumes at statement 1020.

# QUIT

## QUIT (QT)

Clear all breakpoints, close all debugger work files, and disable the debugger trap handler before terminating the debugger task.

### FORMAT:

QT

# SET

## SET

Set values represented by special symbols.

FORMAT:

$$\text{SET } \left\{ \begin{array}{l} \$Bn \\ \$Rn \\ \$P \end{array} \right\} = \left\{ \begin{array}{l} \$Bn \\ \$Rn \\ \$P \\ \text{hex value} \\ \text{dec value} \end{array} \right\} \left\{ \begin{array}{l} (+) \\ (-) \\ * \\ / \end{array} \right\} \left[ \begin{array}{l} \$Bn \\ \$Rn \\ \$P \\ \text{hex value} \\ \text{dec value} \end{array} \right]$$

ARGUMENTS:

$\$Bn$             base register  
 $\$Rn$             index register  
 $\$P$              p-counter  
hex value      hexadecimal value  
dec value      decimal value

DESCRIPTION:

For each set list item, the item on the left is assigned the value of the item on the right. The expression must evaluate to a hexadecimal value when setting base or P-registers, or to a decimal or hexadecimal value when setting index registers.

Example:

```
SET $R1 = %F3E2
```

Register R1 is set to hexadecimal value F3E2.

NOTE

The SET directive is generally not useful to users running only COBOL or FORTRAN programs.

# SLEEP

## SLEEP (SP)

Return processing to command level after initial breakpoints have been set.

### FORMAT:

SP

### DESCRIPTION:

The SP directive temporarily suspends the execution of the debugger and returns control to the Command Processor. You can now start execution of the bound unit in the standard manner. The debugger becomes active again if:

- A breakpoint is reached in the user's program.
- The user types DEBUG from the command level.

# TRACE

## TRACE (OR TR)

Trace the flow of a program. The TRACE directive can also be used in conjunction with the IF directive to monitor the contents of a variable. You must be at a breakpoint to issue the TRACE directive.

### FORMAT:

```
{ TRACE } -OFF [(request_list)]  
{ TR }
```

### ARGUMENTS:

-OFF

Turn tracing off for the current bound unit.

request\_list

Optional list of one or more directives to be performed when a tracepoint is reached. A request list consists of a single directive or a series of directives delimited by parentheses. Directives in a request list are separated by either semicolons or new line characters.

### DESCRIPTION:

A trace is defined and becomes active when the next GO directive is entered. As each statement is executed, a trace message consisting of the line number, bound unit name, and statement label (if any) is listed, the request list (if any) is performed, and execution of the program continues.

Multiple TRACE directives can be entered, but only one request line (the last one entered) will be in effect.

Examples:

TRACE (DUMP TAG)

Print the variable TAG at every statement.

TR (IF J\_INDEX = 0 (PAUSE))

Monitor the contents of J\_INDEX during execution of the current object unit. If J\_INDEX goes to zero, execution pauses.

## MULTIUSER DEBUGGER (SYMBOLIC MODE) PROCEDURES

This subsection guides you through a sample session using the debugger in symbolic mode. You can follow along at a terminal step by step. The original source program could be written in either Advanced FORTRAN or Multiuser COBOL.

### Compiling a Program For Use With the Debugger

Compile your program with either the Advanced FORTRAN or Multiuser COBOL compiler using the `-SYMBOL` argument. The `-SYMBOL` argument creates a symbol table file, `name.Z`, where `name` is the name of your source program. The compiler commands are described next.

#### FORMAT:

```
{ COBOLM } name  -SYMBOL [ctl_arg]
{ FORTRANA }
```

#### ARGUMENTS:

`name`                    Name of your source program.

`ctl_arg`                Other control arguments you wish to use. See the Commands manual for the complete command description.

If necessary, correct any compilation errors and recompile. Proceed with the next step when you have an error-free compilation.

#### SAMPLE COMPILATION DIALOGS

To compile the Advanced FORTRAN program `NFTYPM`, the compilation dialog might be:

```
FORTRANA NFTYPM -SYMBOL
```

Invoke the Advanced FORTRAN compiler specifying that object code be listed, a special symbol table be created, and 4K of memory be used.

```
FORTRANA 2.0 07/09/1302
The compiler is invoked.
000/000 W/E COUNT NFTYPM
```

There are no warnings or errors. Control returns to command level.

```
RDY:
```



To compile the Multiuser COBOL program COMPTV, the compilation dialog might be:

```
COBOLM COMPTV -SYMBOL
```

Invoke the Multiuser COBOL compiler specifying that a special symbol table be created.  
The compiler is invoked.  
There are no errors.  
Control returns to command level.

```
COBOLM x.x 07/15/0813
```

```
NO FATAL ERRORS OR WARNINGS IN COMPTV  
RDY:
```

### Linking an Object Unit With the Debugger

Link the object unit resulting from successful compilation using the `-SYMBOL` option. `-SYMBOL` creates a separate link file named `buname.V`, where `buname` is the name of the bound unit created by the link.

The Linker command's description follows:

**FORMAT:**

```
LINK  buname -SYMBOL [ctl_arg]
```

**ARGUMENTS:**

`buname`                      Name of the bound unit to be created.

`ctl_arg`                      Other control arguments. See the Commands manual for the complete command description.

For more information on the Linker, see Section 6. After you have linked successfully, go on to the next step.

### SAMPLE LINKER DIALOGS

To link object unit NFTYPM, compiled by the Advanced FORTRAN compiler above, the Linker dialog might be:

```
LINKER NFTYPM -SYMBOL -PT
```

Invoke the Linker specifying creation of a special debug link map and a prompt for the user.  
The Linker is invoked.  
You are prompted for a directive.  
Include the standard Advanced FORTRAN runtime library, ZFlRT.

```
LINKER 1982/06/18 0912:50.5  
L?
```

```
LIB >LDD>ZF1RT
```

L?  
LINK NFTYPM

Link the object unit.

L?  
QUIT  
ROOT NFTYPM  
LINK DONE  
RDY:

Terminate the Linker.  
The bound unit is created.  
The Linker is finished.  
Control returns to command level.

To link object unit COMPTV, compiled by the Multiuser COBOL compiler above, the Linker dialog might be:

LINKER COMPTV -SYMBOL -PT

Invoke the Linker, specifying a special debug link map be created and a prompt be given.

LINKER 1982/06/18 0912:50.5  
L?

The Linker is invoked.  
You are prompted for a directive.

LIB >LDD>ZCMRT

Include the standard Multiuser COBOL runtime library, ZCMRT.

L?  
LINK COMPTV

Link the object unit.

L?  
QUIT  
ROOT COMPTV

Terminate the Linker.  
The bound unit is created.  
The Linker is finished.  
Control returns to command level.

LINK DONE  
RDY:

### Invoking the Debugger

To initiate the debugger and set breakpoints in the program to be debugged, issue the Debug command.

FORMAT:

DEBUG buname

ARGUMENTS:

buname

Name of the bound unit to be debugged.

The debugger responds with a prompt (the greater-than sign (>)). Set breakpoints using the AT directive. You can set up to 32 breakpoints. They are numbered from 31 to 0 in descending order. During initialization you can also list the breakpoints with the LIST directive and clear erroneously set breakpoints with the CLEAR directive.

When you are satisfied with the breakpoints you have set, issue the Sleep (SP) directive to temporarily suspend the debugger and return to the command processor. Now you can begin execution of your program with the debugger.

#### SAMPLE INITIALIZATION DIALOG

The debugger initialization dialog for NFTYPM might look like this:

|                                     |   |
|-------------------------------------|---|
| <b>DEBUG NFTYPM</b>                 | Invoke the debugger for the bound unit named NFTYPM.                  |
| >                                   | The debugger responds with its prompt, the greater-than sign.         |
| <b>AT 12</b>                        | Set a breakpoint at line 12.  |
| BP 31 SET                           | The breakpoint id is given in response.                               |
| >                                   |   |
| <b>LIST *</b>                       | List all breakpoints.   |
| BP 31 BU=NFTYPM CU=NFTYPM LINENO=12 | The debugger responds with a list of all current breakpoints by ID.   |
| >                                   |   |
| <b>AT 19</b>                        | Set a breakpoint at line 19.  |
| BP 30 SET                           |   |
| >                                   |   |
| <b>AT 24</b>                        | Set a breakpoint at line 24.  |
| BP 29 SET                           |   |
| >                                   |   |
| <b>CL 29</b>                        | Clear breakpoint 29, at line 24.                                      |
| BP 29 DELETED                       | The debugger acknowledges the deletion.                               |
| >                                   |   |
| <b>AT 25</b>                        | Set a breakpoint at line 25.  |
| BP 29 SET                           |   |
| >                                   |   |
| <b>LIST *</b>                       |   |
| BP 31 BU=NFTYPM CU=NFTYPM LINENO=12 |   |
| BP 30 BU=NFTYPM CU=NFTYPM LINENO=19 |   |
| BP 29 BU=NFTYPM CU=NFTYPM LINENO=25 |   |
| >                                   |   |
| <b>SP</b>                           | Temporarily suspend the debugger and return to the command processor. |

## DEBUGGING MULTIPLE BOUND UNITS

The debugger can be invoked for one bound unit. After that, the debugger must be turned off before other bound units can be debugged. To turn off the debugger, issue the command sequence:

```
DEBUG
QT
```

Then another bound unit can be debugged by re-invoking the debugger with the DEBUG directive and the new bound unit name.

### Executing Your Program With the Debugger

Execute your program by entering the bound unit name. Execution is suspended at the first breakpoint set during debugger initialization. The AT directive allows you to specify a list of debugger directives (a request list) to be executed when the specified breakpoint is reached. If you included a request list when you set the breakpoint, the request list executes. If not, the debugger enters interactive mode and issues the greater-than prompt (>). You can then enter any valid debugger directives to check out the program.

### Sample Executing Dialog

Assume you initialized the debugger using the previous example, your execution then might look like this:

```
NFTYPM
```

```
*BRKPT 31 AT LINE 12
```

```
>
```

```
DUMP YY
```

```
YY 81
```

```
>
```

```
CHANGE YY = %3834
```

```
>
```

```
DUMP YY
```

```
YY 84
```

```
>
```

Begin execution of the program NFTYPM.

Execution stops at line 12, the first breakpoint set.

The debugger prompts for a directive.

Print the current value of variable YY.

The value is printed.

Change the value of YY to ASCII 84 (hexadecimal 3834).

Check that the change was made.

GO

Continue execution. If you found an error, you could issue the Quit (QT) directive to terminate the debugger.

\*BRKPT 30 AT LINE 19

Execution resumes until the next breakpoint is encountered.

>

The debugger prompts for a directive.

TRACE (IF YY = 81 (PAUSE))

Trace the value of YY and terminate if YY is not equal to 81.

RDY:

YY is not equal to 81, so the program terminates and you return to the command level.



REMOVE THIS PAGE AND PLACE TAB FOR

TAB 8

MULTIUSER D.B. NUM.





## *Section 8*

# **MULTIUSER DEBUGGER (NUMERIC MODE)**

The Multiuser Debugger is a general purpose tool used for testing application programs. The debugger operates in two modes:

- Numeric Mode--primarily used for applications written in Assembly language (can be used on any program)
- Symbolic Mode--used for applications written in higher-level languages such as COBOL or FORTRAN.

Symbolic debugging is described in Section 7 of this manual. Numeric debugging is described in this section; full information on numeric debug functions and directives is provided. Procedural information on using numeric debugging directives to debug a bound unit is also included in this section.

## OVERVIEW

The Multiuser Debugger is an interactive tool used at program execution time to debug bound units. In numeric mode, it can debug COBOL, FORTRAN, BASIC, and Assembly language programs. This facility runs in a protected environment and is available to multiple users; it can be used by all task groups (except \$D) running on the system. For information on \$D, see Section 9.

## CAPABILITIES

In numeric mode, data is referred to by program locations in terms of memory (segment) addresses. Data is displayed in hexadecimal or hex-ASCII dump format. Using Multiuser Debugger commands, you can suspend programs at selected breakpoints during execution, restart at the same point, or change sequence and start from a different point. While the program is suspended, you can examine input, display data, or alter values.

With the Multiuser Debugger you can:

- Define, store, and execute a sequence of directives
- Set or clear true breakpoints in task code to monitor task status
- Set or clear bound unit breakpoints to gain control of bound units as they are loaded
- Set or clear quick breakpoints (from the \$\$ group only) to monitor time-dependent tasks without undue distortion of time
- Display, change, and dump either memory or registers
- Evaluate expressions.

### NOTE

All concurrent users of the multiuser debugger must use a common copy of the DEBUG bound unit. Attempting to use a different copy of DEBUG from the one currently being used will result in a 5D22 error. For this reason, you should not make a copy of DEBUG.

## INVOKING THE DEBUGGER (NUMERIC MODE)

The command used to invoke the Multiuser Debugger is:

DEBUG

There are no valid arguments with this command.

## DEBUGGER FILE REQUIREMENTS

For true and bound unit breakpoints, Debugger directives can be stored in a user-defined work file. If used, this file must be opened by the Specify File directive (the SF directive is described later in this section) and must always be followed by the suffix ".DB". This work file requires a size of 64 sectors on any media.

The Multiuser Debugger directives associated with quick breakpoints are stored in memory, and optionally, in a work file as described above. Output generated by these directives is written to memory and, optionally, to a user-defined quick disk file. This quick disk file must have been created previously outside the Debugger task using the Create File command (see the Commands manual for details) and must be referenced within the Multiuser Debugger task by a Specify File (SF) directive. This file must end with the suffix ".QK".

The Debugger directives mentioned above are identified and described in Table 8-2, later in this section.

## DEBUGGER MEMORY REQUIREMENTS

To set true or bound unit breakpoints, the reentrant portion of the Multiuser Debugger requires a minimum memory area of 2784<sub>10</sub> words. The separate data portion of the amount of memory required per group is approximately 1984<sub>10</sub> words. This includes all non-reentrant code, the Multiuser Debugger overlay area, and all necessary data information.

To debug time-dependent tasks using quick breakpoints, the total amount of memory required is 8472<sub>10</sub> words (for reentrant and data portions) plus the amount of memory you requested for the quick memory buffers. The quick memory buffers are described later in this section.

## DEBUGGER OPERATION

The Multiuser Debugger is restricted to the write privileges of the group it serves; several users can debug within their own groups without affecting other groups. Since the Multiuser Debugger runs under any user-defined group, memory protection is dependent upon the task group. If no memory protection is established, you can alter any and all memory; therefore, the task should run in a protected environment.

The Multiuser Debugger handles traps to trap vector 06 (register overflow), trap vector 14 (unauthorized reference to protected memory), trap vector 15 (reference to unavailable resource), and trap vector 34 (segment fault) and continues as described below.

An error message is displayed if you try to access nonvirtual memory within any Multiuser Debugger directive except the Dump Memory (DP) directive. If a Trap-to-Trap-Vector-15 occurs when a DP directive is specified, the Multiuser Debugger dumps as much of the requested memory as possible. Once a nonvirtual address is invoked, the rest of the current line to be printed is blank-filled. The current nonvirtual address is advanced to the value that is the next multiple of 1K. The procedure continues until the area to be dumped is exhausted or the end of memory is reached.

### ENTERING DIRECTIVES

Multiuser Debugger directives consist of a directive name only or a directive name and one or more arguments. Within a directive, arguments are separated from each other by one or more spaces. Multiple Debugger directives can be entered on a single line; each directive, except the last, must be followed by a semicolon (;). At the end of each line (i.e., immediately after the last or only directive), press carriage return. Except where otherwise specified, all argument values are entered in hexadecimal notation.

Debugger directives may only be entered when the Debugger has control of the group. This occurs when:

- The Debugger is loaded.
- A breakpoint occurs.
- You press the BREAK key and DEBUG is typed as the post-break input. (BREAK key functionality is described later in this section.)

Table 8-1 summarizes Multiuser Debugger directives by function. These directives are described in detail on the following pages.

#### NOTE

Pay careful attention to the format of each directive, because the use of delimiters, if any, between a directive name and the first (or only) argument varies according to which directive is being specified.

Special symbols are used in the Multiuser Debugger directive lines. These symbols are described in Table 8-2.

#### NOTE

The Multiuser Debugger only recognizes tasks that are in a trapped state.

Table 8-1. Summary of Numeric Mode Directives

| Function                      | Directive              | Directive Name  |
|-------------------------------|------------------------|---|
| Directive line and handling   | Dn                     | Define directive line n   |
|                               | En                     | Execute directive line n  |
|                               | P*                     | Print all predefined directive lines                            |
|                               | Pn                     | Print predefined directive line n                               |
| True breakpoint               | C*                     | Clear all true breakpoints                                      |
|                               | Cn                     | Clear true breakpoint n   |
|                               | L*                     | List all true breakpoints and associated directive lines        |
|                               | Ln                     | List true breakpoint n and associated directive line            |
|                               | Sn                     | Set true breakpoint n   |
| Bound unit breakpoint control | CB*                    | Clear all bound unit breakpoints                                |
|                               | CBn                    | Clear bound unit breakpoint n                                   |
|                               | LB*                    | List all bound unit breakpoints and associated directive line   |
|                               | LBn                    | List bound unit breakpoint n and associated directive line      |
|                               | SBn                    | Set bound unit breakpoint n                                     |
| Quick breakpoint control      | XBn                    | Set express bound unit breakpoint n                             |
|                               | CQn                    | Clear quick breakpoint n  |
|                               | CQ*                    | Clear all quick breakpoints                                     |
|                               | LQn                    | List quick breakpoint and its associated directive line         |
|                               | LQ*                    | List all quick breakpoints and their associated directive lines |
|                               | MQ                     | Get memory block for quick breakpoint information storage       |
|                               | PQ                     | Print pointer to quick memory block                             |
|                               | RQ                     | Return quick memory   |
| SQn                           | Set quick breakpoint n |   |
| Trace trap control            | DT                     | Define trace directive line                                     |
|                               | PT                     | Print trace directive line                                      |
|                               | ST                     | Start j-mode trace  |
|                               | ET                     | End j-mode trace  |
| Memory and register control   | AR                     | Print contents of all registers of the active level             |
|                               | CH                     | Change memory   |
|                               | DH                     | Display memory in hexadecimal                                   |
|                               | DP                     | Dump memory in hexadecimal and ASCII                            |

\*

Table 8-1 (cont). Summary of Numeric Mode Directives

| Function              | Directive | Directive Name  |
|-----------------------|-----------|---|
| Symbol control        | AS        | Assign a hexadecimal value to symbol  |
|                       | VH        | Print value of expression in hexadecimal  |
| General execution     | E         | Temporary escape to the command processor   |
|                       | FI        | Redirect input  |
|                       | FO        | Redirect output   |
|                       | GO        | Continue execution from breakpoint  |
|                       | Hn        | Print header line   |
|                       | IF        | Conditional execution   |
|                       | MODE      | Change from numeric to symbolic mode or vice-versa  |
|                       | RF        | Reset file location   |
| Abnormal Trap Control | SF        | Specify file location   |
|                       | SP        | Temporarily suspend the Multiuser Debugger; return control to the command processor (sleep) |
|                       | QT        | Abort Multiuser Debugger task (quit)  |
| Abnormal Trap Control | CT        | Clear abnormal trap bit   |
|                       | TB        | Turn on abnormal trap bit   |
|                       | TT        | Terminate trapped task  |

NOTE

The memory and register control directives (AR, CH, DH, and DP) apply to registers on the active level. To determine which level is the active level and/or to set the active level to a specified value, see "Determining/Setting the Active Level" below.

Table 8-2. Symbols Used in Numeric Mode Directive Lines

| Symbol Type                 | Meaning  |
|-----------------------------|--|
| <u>Arithmetic Operators</u> |  |
| plus sign (+)               | Performs addition.   |
| minus sign (-)              | Performs subtraction.  |
| K                           | Multiplies a hexadecimal integer by 1024 decimal (400 in hexadecimal) when K is the last character of an integer expression. |

Table 8-2 (cont). Symbols Used in Numeric Mode Directive Lines

| Symbol Type              | Meaning  |
|--------------------------|--|
| <u>Address Operators</u> |  |
| period (.)               | Represents the last start address used in a previous memory reference directive (DH, CH, DP).  |
| ampersand (&)            | Represents the address of the next location beyond the last one used by a previous memory reference directive (DH, CH, DP).                                |
| brackets []              | Signifies the contents of the location defined by the expression within the brackets. Three levels of nesting may be used.                                 |
| <u>Reserved Symbols</u>  |  |
| \$Bn                     | Contents of the base register n of the active level. The values 1 through 7 can be used for n.   |
| \$Rn                     | Contents of the data register n of the active level. The values 1 through 7 can be used for n.   |
| \$P                      | Contents of the program counter of the active level.   |
| \$I                      | Contents of the indicator register of the active level.  |
| \$IV                     | Address of the Task Control Block of the task which is currently trapped (i.e., on a true or bound unit breakpoint, a true trap, or a user trap).          |
| \$IV_Bn                  | Represents the address of the base register n storage area in the Interrupt Save Area (ISA) of the active level. The values 1 through 7 can be used for n. |
| \$IV_Rn                  | Represents the address of the data register n storage area in the ISA of the active level. The values 1 through 7 can be used for n.                       |

Table 8-2 (cont). Symbols Used in Numeric Mode Directive Lines

| Symbol Type         | Meaning  |
|---------------------|--|
| \$IV_Mn             | Represents the address of the Commercial or Scientific Instruction Processor mode control register n storage area in the ISA of the active level. For the Commercial Instruction Processor mode, n must equal 3. For the Scientific Instruction Processor mode n can be either 4 or 5. |
| \$IV_Sn             | Represents the address of the scientific accumulator register n storage area in the ISA of the active level. The values 1 through 3 can be used for n.   |
| \$IV_Kn             | Represents the address of the K register n storage area in the interrupt save area (ISA) of the active level. The value of n can be 1 through 7.   |
| \$Kn                | Contents of the K register n of the active level. The value of n can be 1 through 7.   |
| \$S                 | Contents of the system status register (level number and privilege bit only) of the active level.  |
| \$SL                | Represents the value of the level number of the active level.  |
| \$E                 | Represents the entry point of a bound unit as defined in the bound unit or by the caller. This reserved symbol is used only at the time of a bound unit breakpoint, in place of \$P associated with true and quick breakpoints.  |
| \$T                 | Represents the address of the stack of the active level.   |
| G through Z         | Twenty single-character symbols having initial values of zero. Values may be assigned using the AS directive.  |
| \$V_G through \$V_Z | Represents the address of symbolic variable G through Z. This allows variables to be tested by the IF directive, which requires an address for its left argument.  |
| \$CI                | Represents the contents of the Commercial Processor indicator word of the active level.  |



Table 8-2 (cont). Symbols Used in Numeric Mode Directive Lines

| Symbol Type                              | Meaning   |
|--|---|
| \$C1                                     | Represents the contents of the Commercial Processor remote descriptor table of the active level.  |
| \$SI                                     | Represents the contents of the Scientific Instruction Processor (SIP) indicator word of the active level.   |
| \$Mn                                     | Represents the contents of the mode control register of the active level. The values 1 through 7 can be used for n.   |
| Debug Language:<br><br>^ <<br>^ =<br>^ > | The condition to be satisfied in an IF directive for continuous processing of the directive line. "^" indicates a logical NOT which may optionally be used.   |
| parentheses ( )                          | Indicates directive or header information to be stored for later use. An unmatched right parenthesis results in an error. A right parenthesis that is paired with the first left parenthesis terminates the directive definition.   |
| exp                                      | Indicates a valid expression formed by using expression elements. Expression elements are addresses, reserved symbols, and hexadecimal values up to 32 bits in length. No more than one address is allowed within an expression. An expression element may be preceded by the positive (+) or negative (-) unary operator. Expression elements can be joined by the addition (+) or subtraction (-) operator. |
| rexp                                     | Consists of exp /exp , where exp is a hexadecimal number that is a value of a location expression; exp is an optional hexadecimal repeat factor whose value must be between 1 and 32,767. If exp is omitted, there is no repetition.  |
| ;  | Separation character between directives on the same line.   |
| *  | Signifies "all" in certain Print, Clear, and List directives.   |

## DEBUGGER AND BREAK KEY FUNCTIONALITY

Typing DEBUG as a response to the BREAK key transfers you to the Multiuser Debugger task. To return to the previous stack level, enter the Sleep (SP) directive or terminate the Debugger completely with the Quit (QT) directive. The description of the Debugger and BREAK key functionality applies only to true and bound unit breakpoints, and not to quick breakpoints. BREAK key functionality is not supported in the \$S task group.

If DEBUG is the task that was broken, any command is a valid response, including PI, UW, SR, or NEW\_PROC.

### NOTES

1. The Program Interrupt (PI) response returns the user group to the Debugger input level and allows the entry of Debugger directives.
2. If the Debugger task was broken and DEBUG is entered as the response, you are placed in the Debugger input mode.
3. The Unwind (UW) response causes the Debugger to execute either the GO or SP directive, depending on which is appropriate at the time of the **\*\*BREAK\*\***. If the Debugger was activated as the result of encountering a breakpoint, entering UW causes execution of the GO directive.

## PLANNING CONSIDERATIONS

### Setting True Breakpoints and Bound Unit Breakpoints

True breakpoints and bound unit breakpoints can be set to trap at selected task code locations. At true breakpoints, memory and register values can be displayed and changed. At bound unit breakpoints, only memory can be displayed and changed. The registers displayed at the time of a bound unit breakpoint are not those of the trapped task. In this way, a task can be executed, the value of its variables checked as execution proceeds, code modified and, if necessary, variable values changed in order to test the sequence of code up to the next breakpoint.

Express bound unit breakpoints allow you to cause a true breakpoint to be set at bound unit load time, avoiding the inconvenience of having to stop at a bound unit breakpoint and setting the first true breakpoint.

## Setting Global Breakpoints

Any true, bound unit, or express bound unit breakpoint set in the system task group (\$S) will be activated regardless of which task group encounters it. For this reason, these breakpoints are called global breakpoints.

Certain effects need to be considered when setting either bound unit breakpoints or express bound unit breakpoints in group \$S. If normal bound unit breakpoints are to be used in group \$S, the command processor must be loaded (e.g., EC !CONSOLE). (This is done to prevent a system hang.) If express bound unit breakpoints are to be used in group \$S, the command processor must not be loaded. It is recommended that you use only express bound unit breakpoints as global breakpoints without the command processor.

When a global breakpoint is encountered for another group, the group is identified by an output line preceding the breakpoint banner line. The format is:

For Group id:

where id is the two-character group id number.

## Setting Quick Breakpoints

Quick breakpoints can be set to trap at selected locations to monitor time-dependent functions (for example, monitoring a driver). At these breakpoints, memory and registers can be stored in a block of memory (reserved by means of the Get Quick Memory directive) and, optionally, in a disk file to be retrieved and studied at some later time at your convenience. These breakpoints must be set when you are running in the system task group (\$S).

## Preliminary Steps for Using Quick Breakpoints

Before invoking the Debugger from the \$S task group:

1. Calculate the approximate amount of memory necessary for the quick memory buffers.
2. Create a Debugger quick disk file with the format

path.QK

using the Create File (CR) command (see the Commands manual). The quick disk file must be created from a user-defined group. It should be created as a relative file with a control interval (CI) size greater than or equal to the size of the quick memory blocks that you specify in the Get Quick Memory (MQ) directive.

### 3. Enter the command

EC !CONSOLE

to load the command processor.

Now you can invoke the Multiuser Debugger and monitor the time-dependent task without causing any time distortion within the task.

### Guidelines for Setting Breakpoints

The following guidelines should be observed in setting breakpoints:

- True breakpoints can be set in a bound unit in a task group (or in an overlay of a bound unit in a task group) only when the task group/overlay currently is memory resident. Use the SBn (Set Bound Unit Breakpoint) directive to gain control of a task group bound unit/overlay when it is loaded, to allow true breakpoints to be properly set.
- True breakpoints may not be set in code that will be executed at the inhibit level (level 3).
- True breakpoints are set in task groups by specifying the Set Breakpoint (Sn) directive. (The detailed description of the Sn directive later in this section includes additional rules for specifying true breakpoints.)
- Quick breakpoints can only be set from the system task group (\$S); i.e., you must be debugging from the terminal designated as the operator terminal.
- Quick breakpoints are set in the \$S task group by specifying the Set Quick Breakpoint (SQn) directive. (The detailed description of the SQn directive later in this section includes additional rules for specifying quick breakpoints.)
- Only quick breakpoints may be set in shareable code.
- Quick breakpoints may be embedded in true or bound unit directive lines. Note that in this case you set all breakpoints from the system task group and that these breakpoints could impact all users. Thus, caution must be used when debugging in the system task group.

### Controlling Output

Output can be redirected by using a true or bound unit breakpoint. When the breakpoint condition occurs, the FO directive can be used to redirect the output.

When quick breakpoints are utilized, output sent to the previously specified user-defined disk file can be retrieved after closing the disk file and, outside the Multiuser Debugger task, entering the PR\_QK command (see the Commands manual for details.)

### Determining the Active Level

\*

The active level is the priority level currently in effect. When the Debugger is activated by a breakpoint, trace trap, or user trap, the active level is automatically set. The active level can be displayed by displaying the pseudo-variable \$SL (i.e., 'VH \$SL').

### Maintaining a Trace History

When using the Debugger with disk-stored directive lines that execute upon encountering a trap or a breakpoint, a trace history may be maintained on the device specified as user-out.

Also, while at a Debugger breakpoint, the suspended task may be set to run in jump-trace mode (j-mode). In this case, every departure from the current sequence of instructions generates a trace trap.

### DEBUGGER DIRECTIVES

The rest of this section consists of detailed descriptions of the Multiuser Debugger directives, presented in alphabetic order.

The following notational symbols are used to describe the format of Multiuser Debugger directives.

#### Notational Symbols

#### Meaning

|                    |   |
|--------------------|---|
| braces { }         | For a single enclosed argument, indicates that the argument is optional. If more than one argument is enclosed by braces in a vertical listing, the braces indicate that a choice is to be made. In this case, optional arguments are identified in the text. |
| Ellipsis (...)     | Indicates the ability to repeat within braces.  |
| Delta ( $\Delta$ ) | Indicates one or more spaces.   |
| Vertical bar ( )   | Indicates a choice between two or more arguments.   |

Note that the use of braces shown above differs from the usage defined in the preface and employed in other sections.

# ALL REGISTERS

## ALL REGISTERS (AR)

The All Registers (AR) directive prints all registers for the active level on the device specified as user-out. Bound unit breakpoints lie within the loader, not in the task context. As a result, the display of registers at a bound unit breakpoint are not those of the task and can be ignored.

### FORMAT:

AR

# ASSIGN

## ASSIGN (AS)

The Assign (AS) directive assigns a specified hexadecimal value to a specified symbol; this directive alters registers of the active level, and defines reserved symbols. Bound unit breakpoints lie within the loader, not in your task context. As a result, the Assign directive on a register is refused by the Multiuser Debugger, if the current level's task is suspended on a bound unit breakpoint.

### FORMAT:

```
AS sym exp {sym exp...}
```

### ARGUMENTS:

sym

A reserved symbol G through Z or a register.

exp

An expression that resolves to a hexadecimal value up to 32 bits. The rightmost 20 bits are used for an address register (\$Bn), the program counter (\$P), or the bound unit entry point (\$E); the rightmost 16 bits are used for all other registers.

### Example:

```
AS $R1 -2 X 1408 $B7 X+15
```

-2 is assigned to data register 1, 1408 is assigned to the reserved symbol X, and 141D assigned to base register 7.

# CHANGE MEMORY

## CHANGE MEMORY (CH)

The Change Memory (CH) directive changes the contents of a single specified memory location, or consecutive locations starting at that location, to specified value(s).

### NOTE

This directive changes memory only. To alter register contents, see the Assign (AS) directive.

### FORMAT:

CH exp rexp {rexp...}

### ARGUMENTS:

exp

First or only location whose contents are to be changed.

rexp

Value(s) to be put in memory location(s).

### Example 1:

CH 200 4FFF 1716

Put the value 4FFF into location 200 and 1716 into location 201.

### Example 2:

CH 100 0/10

Locations 100 to 10F are zero-filled.

### Example 3:

CH 2000 0/10 1/10 2/10

This example shows how multiple repeat factors can be used: Locations 2000 to 200F are given a value of zero, locations 2010 to 201F are given a value of 1, and locations 2020 to 202F are filled with 2s.



## CLEAR ABNORMAL TRAP BIT

### CLEAR ABNORMAL TRAP BIT (CT)

Clear the abnormal trap bit set in the debugger's indicator word.

This bit is set to request that a special debug breakpoint message be displayed if a task in a group encounters an unexpected (abnormal) 0303xx trap condition. If the bit is not set, the trap information is displayed and the task is terminated.

With the bit set, the trap information is displayed, the task is suspended, and a special breakpoint message appears. These events allow the user to decide whether to continue executing the task (by entering GO) or to terminate the task (by entering TT).

FORMAT:

CT

# CLEAR ALL BOUND UNIT BREAKPOINTS

## CLEAR ALL BOUND UNIT BREAKPOINTS (CB\*)

The Clear All Bound Unit Breakpoints (CB\*) directive clears all bound unit breakpoints, but not their associated directive lines.

FORMAT:

CB\*

## CLEAR ALL QUICK BREAKPOINTS

### CLEAR ALL QUICK BREAKPOINTS (CQ\*)

The Clear All Quick Breakpoints (CQ\*) directive clears all quick breakpoints, but not their associated directive lines.

FORMAT:

CQ\*

# **CLEAR ALL TRUE BREAKPOINTS**

## CLEAR ALL TRUE BREAKPOINTS (C\*)

The Clear All True Breakpoints (C\*) directive clears all defined true breakpoints, but not their associated directive lines.

FORMAT:

C\*

## CLEAR BOUND UNIT BREAKPOINT

### CLEAR BOUND UNIT BREAKPOINT (CBn)

The Clear Bound Unit Breakpoint (CBn) directive clears a specified breakpoint for a bound unit, but does not clear the associated directive line.

FORMAT:

CBn

ARGUMENT:

n

Specifies the bound unit breakpoint to be cleared; must be a decimal digit from 0 to 9.

Example:

CB3

Breakpoint number 3 is cleared for the bound unit previously defined by SB3; the associated directive line is not cleared.

# CLEAR QUICK BREAKPOINT

## CLEAR QUICK BREAKPOINT (CQn)

The Clear Quick Breakpoint (CQn) directive clears a specified quick breakpoint, but not the associated directive line.

### FORMAT:

CQn

### ARGUMENT:

n

Number of the quick breakpoint; must be a decimal digit from 0 through 9.

### Example:

CQ3

Quick breakpoint number 3 is cleared; the associated directive line is not cleared.

# CLEAR TRUE BREAKPOINT

## CLEAR TRUE BREAKPOINT (Cn)

The Clear True Breakpoint (Cn) directive clears a specified true breakpoint, but not the associated directive line.

### FORMAT:

Cn

### ARGUMENT:

n

Number of the true breakpoint; must be from 0 through 31 (decimal).

### Example:

C3

True breakpoint number 3 is cleared; the associated directive line is not cleared.

# CONDITIONAL EXECUTION

## CONDITIONAL EXECUTION (IF)

The Conditional Execution (IF) directive allows a set of conditions to be tested prior to execution of other Multiuser Debugger directives. The IF directive is intended to be used in a stored breakpoint directive line. It permits breakpoints to be reported without suspending the active level if the specified condition does not exist. When a breakpoint occurs for which an IF directive has been specified, the following actions occur:

- Any directives preceding IF are executed.
- The IF conditions are evaluated, as follows:

If TRUE, a line in the following format is displayed on the current Debugger output device

"exp { ^ } { < } { = } { > } { , } hhhh..."

and any directives following IF are executed. If a GO directive does not follow, the active level is suspended.

If FALSE, no display occurs, and the directives following IF are not executed. The active level continues processing.

### FORMAT:

IF exp { ^ } { < } { = } { > } { , } hhhh...;

### ARGUMENTS:

exp

Hexadecimal memory address of a byte string argument. This must specify an address; \$Rn (where 0 < n < 7) cannot be used for exp. No check for this error is performed; however, if you use \$Rn, results are unpredictable.

Symbolic variable address \$V G through \$V Z can be used to represent the address of symbolic variables G through Z.



{^}{<}

Specifies the condition to be tested when comparing the memory byte string value to the test parameter. "^" optionally specifies logical negation; i.e., not less than, not equal, not greater than.

{'}

Indicates that the argument is right-byte aligned.

hhhh...

The test parameter, expressed in ASCII as a string of pairs of hexadecimal digits; each pair represents one byte. The test parameter may not be an assigned symbol (see the Assign (AS) directive). The length of the parameter is limited by the maximum size of a Multiuser Debugger stored directive (127 bytes). The parameter's ASCII value must consist of pairs of hexadecimal values. If an odd number of hexadecimal values is specified, a command error is reported when the directive is executed and the task remains suspended to allow for correction. If the IF directive is embedded in a Quick Breakpoint directive line, this error condition is a false state and the rest of the directive line is ignored and the task continues. The IF directive terminator must be a semicolon (;).

Example:

Assume that true breakpoint 2, as defined below, is encountered, and that \$B7 points to memory location 555F:

S2 135E (IF 1000^>,3E;IF \$B7=42D1;DP \$B7/100;GO)

Two conditions must be true before the Dump (DP) directive is executed:

1. The rightmost byte at memory location 1000 must be less than or equal to 3E.
2. The byte string found at memory location 555F must be equal to 42D1.

## CONDITIONAL EXECUTION

If both conditions are met, the dump is executed, and the active level continues in response to the GO directive. If either condition is not satisfied, the dump does not occur, and the active level continues without suspension.

### NOTE

The IF directive can be entered from the terminal, in which case its action corresponds to its entry in a stored directive line. However, using the IF directive from the terminal is of limited usefulness, since the conditions to be tested can be checked by using other directives (e.g., DH).

# DEFINE

## DEFINE (Dn)

The Define (Dn) directive defines a specified directive line for future use and associates that line with a specified number. The directive line is stored on the user-defined work file and can be referred to by specifying in an Execute (En) directive the number with which it was associated. The entire Define directive may comprise a maximum of 126 characters.

When you reuse a disk that has predefined directive lines from a previous execution, the lines may be referred to without redefining them. (See "Set True Breakpoint (Sn)" later in this section.) This prevents complex predefined directive lines from being respecified each time the system is reloaded for debugging the same problem.

### FORMAT:

Dn (directive line)

### ARGUMENTS:

n

Number with which the specified directive line is associated; must be from 0 through 9.

(directive line)

One or more directives stored for future use.

### Example 1:

D3 (CH 100 0)

Associate the number 3 with the directive within the parentheses. Hereafter, each time the directive E3 (see "Execute (En)".) is executed, the parenthetical directive is executed and location 100 is zero-filled.

### Example 2:

D4 ( )

By storing a null directive, deactivate a previously defined directive line 4 which is no longer required.

# DEFINE TRACE

## DEFINE TRACE (DT)

The Define Trace (DT) directive associates the directive line within the parentheses with the occurrence of a jump trace trap or a BRK instruction not already defined as a breakpoint. The specified directive line is stored in the user-defined work file for future use. The entire Define Trace directive may comprise a maximum of 126 characters.

When you reuse a disk file that has predefined directive lines from a previous execution, the lines may be referred to without redefining them. (See "Set True Breakpoint (Sn).")

### FORMAT:

DT (directive line)

### ARGUMENT:

(directive line)

One or more directives stored for future use.

### Example 1:

DT (AR)

All registers are displayed each time a trace trap occurs.  
(See "All Registers (AR).")

### Example 2:

DT ( )

Cancel usages of the predefined trace directive line.

# DISPLAY MEMORY

## DISPLAY MEMORY (DH)

The Display Memory (DH) directive displays one or more specified memory location(s) in hexadecimal notation either on the terminal or on another specified device.

### FORMAT:

```
DH rexp {rexp...}
```

### ARGUMENT:

rexp

Location(s) whose contents are displayed. A minimum of one location may be displayed.

### Example 1:

```
DH 200
```

Display the contents of location 200.

### Example 2:

```
DH 200/100
```

Display the contents of locations 200 to 2FF.

# DUMP MEMORY

## DUMP MEMORY (DP)

The Dump Memory (DP) directive prints on the terminal or another specified device an area of memory starting at a specified location. The printout comprises a minimum of eight locations and is in hexadecimal and ASCII notations.

### NOTE

Up to 32K words of memory can be dumped in response to a single DP directive. Dumps of more than 32K must be performed as separate operations.

### FORMAT:

DP rexp {rexp...}

### ARGUMENT:

rexp

Memory location(s) whose contents are displayed. The display is always in a multiple of eight locations.

### Example 1:

DP 200

Display (at the current user-out device) one line of memory in both hexadecimal and ASCII, starting at location 200.

### Example 2:

DP 80/3C 200/240

Display the contents of locations 80 to BF and 200 to 43F on the current user-out device. Although the repeat expression of 3C was specified in the directive, the display is through location BF because displays are always in multiples of eight locations.

## END TRACE

### END TRACE (ET)

The End Trace (ET) directive disables the j-mode trace (see "Start J-Mode Trace (ST)") for a specific task on the next trap.

#### FORMAT:

ET lvl

#### ARGUMENT:

lvl

The level, as previously specified by the last ST directive. lvl is preceded by one space. The trace must first have been enabled using ST.

# ESCAPE

## ESCAPE (E)

The Escape (E) directive passes the rest of the input buffer to the command processor for processing. Debugger directives can precede the portion of the input buffer to be passed to the command processor, if they are separated by semicolons (;). They cannot, however, follow commands passed to the command processor. Once an Escape directive has been encountered, the rest of the input line is interpreted by the command processor. This allows multiple commands to be passed to the command processor using only one Escape directive.

### FORMAT:

E exp{;exp}

### ARGUMENT:

exp

Any command.

### Example:

E TIME

Return the time.

### NOTE

Do not use the Escape directive to invoke the bound units that you intend to debug. The Multiuser Debugger must be terminated (see "Sleep (SP)" for details) before invoking a bound unit containing breakpoints.



## EXECUTE

### EXECUTE (En)

The Execute (En) directive retrieves and executes a specified predefined directive line. This directive may not be embedded in Define (Dn) directive lines; it is permitted in Set True Breakpoint (Sn), Define Trace (DT), and Set Bound Unit Breakpoint (SBn) lines.

#### FORMAT:

En

#### ARGUMENT:

n

Number of the line to be executed; must be from 0 through 9.

#### Example 1:

```
D3 (CH 100 0)
E3
```

The directive E3 retrieves and executes line 3, previously defined in the Define directive as CH 100 0.

#### Example 2:

```
D3 (CH 100 0)
S1 100 (E3)
```

The Execute (E3) directive is embedded in a Set True Breakpoint directive line. The Execute directive retrieves and executes line 3, previously defined in the Define directive as CH 100 0, whenever true breakpoint 1 is encountered.

# FILE IN

## FILE IN (FI)

The File In (FI) directive causes input directives to be taken from the file named by the path argument. The directive file must end with an FI directive to switch back to the user-in file.

### FORMAT:

FI {path}

### ARGUMENT:

{path}

The pathname of the file or device from which subsequent directives are to be taken.

### Example:

FI MY.FI

Input is taken from the file MY.FI until another FI directive is read. If an EOF is reached on the input file (no FI directive to redirect input), DEBUG terminates with an error message.

## FILE OUT

### FILE OUT (FO)

The File Out (FO) directive redirects output from the current user-out file to the device specified by the pathname argument. This directive allows messages that result when a true or bound unit breakpoint or other condition occurs to be sent to a device other than the user-out file. It has no effect on input to the program.

#### FORMAT:

FO {path}

#### ARGUMENT:

{path}

The pathname of the device to which output for the group is directed. If path is omitted, user-out defaults to the group's original user-out file.

#### Example:

FO !LPT00

Output is redirected from the current user-out file to a line printer.

# GET QUICK MEMORY

## GET QUICK MEMORY (MQ)

The Get Quick Memory (MQ) directive reserves the requested amount of memory for storing (in memory buffers) the output from execution of a Quick Breakpoint directive line.

### FORMAT:

MQ {-BS exp} {-RS exp} {-NB exp}

### ARGUMENTS:

{-BS exp}

Size of buffer specified in words (hexadecimal).  
Default: 800

{-RS exp}

Size of record specified in words (hexadecimal).  
Default: 100

{-NB exp}

Number of buffers requested; must be two or more.  
Default: 2

### NOTES

1. To use quick breakpoints, enter this directive first after invoking the Multiuser Debugger in the \$S task group.
2. Each buffer must contain at least two records. The first record of each buffer contains the information needed by the Multiuser Debugger as listed below.

| <u>Offset</u> | <u>Number<br/>of Words</u> | <u>Definition</u>                           |
|---------------|----------------------------|---|
| 0             | 4                          | File system information                     |
| 4             | 1                          | Quick file identifier                       |
| 5             | 2                          | Pointer to next buffer in chain             |
| 7             | 2                          | Pointer to next available record in buffer  |
| 9             | 1                          | Maximum record size                         |
| 10            | 1                          | Number of records still available in buffer |
| 11            | 1                          | Number of records not completed in buffer   |
| 12            | 1                          | Number of records for reset                 |
| 13            | 1                          | Current buffer number (n+1)                 |
| 14            | 1                          | Number of buffers in the memory block       |
| 15            | 1                          | Indicators word                             |
|               |                            | X'1' buffer in use                          |
|               |                            | X'2' buffer full                            |
|               |                            | X'4' buffer ready for reuse                 |
|               |                            | X'8' buffer information lost                |
|               |                            | X'10' last buffer to be written to disk     |
|               |                            | X'20' disk file has cycled                  |

The minimum size for each record is 19 words.

Example:

MQ -RS 80

Request memory. The default values for buffer size and the number of buffers requested are used. Each record is 80 words long.

# GO

## GO

The GO directive resumes execution on the current active level after a breakpoint and can optionally specify a limit-to-pause counter value which applies only to j-mode trace traps. (See "Start J-Mode Trace (ST)").

### FORMAT:

GO {LLLL}

### ARGUMENT:

{LLLL}

Optionally, an ASCII expression of 1 to 4 hexadecimal digits greater than zero. The ASCII expression is preceded by one space. Default: 1

### Example:

S0 100 (DH 200/10;GO)

The task encountering true breakpoint 0 traps; the Associated directive line is executed by the Multiuser Debugger and the last directive of the directive line (GO) reactivates the task.

## LIST ALL BOUND UNIT BREAKPOINTS

### LIST ALL BOUND UNIT BREAKPOINTS (LB\*)

The List All Bound Unit Breakpoints (LB\*) directive displays all bound unit breakpoints and their associated directive lines if the work file is open. If the work file is not open, only defined bound unit breakpoints are listed. If the work file is open, the listing contains all bound unit breakpoints and all associated directive lines. It is possible to list a bound unit breakpoint with no corresponding directive line or a directive line with no defined bound unit breakpoint. However, if neither a bound unit breakpoint nor a directive line is defined for a particular bound unit breakpoint number, that bound unit breakpoint number does not appear in the list.

#### FORMAT:

LB\*

#### Sample Listing:

```
BU0      (S0  $E;GO)
BU2 LWD ( )
```

The work file is open and bound unit 0 has a directive line but no defined breakpoint; bound units 1 and 3 through 9 have neither defined breakpoints nor directive lines; and bound unit 2 has only a defined breakpoint.

#### NOTE

Ten bound unit breakpoints (one per bound unit; 0 through 9) can be set. (See "Set Bound Unit Breakpoint (SBn)".)

If a bound unit breakpoint is in express mode, the output of the LB\* directive contains an additional field in the format:

:nnnn

where nnnn is the hexadecimal offset specified for the resulting true breakpoint.

# LIST ALL QUICK BREAKPOINTS

## LIST ALL QUICK BREAKPOINTS (LQ\*)

The List All Quick Breakpoints (LQ\*) directive displays all quick breakpoints and their associated directive lines. You can print a directive line without an associated quick breakpoint (e.g., if the quick breakpoint has been previously cleared). If neither a quick breakpoint nor a quick breakpoint directive line is defined for a particular quick breakpoint number, that breakpoint does not appear in the list.

### FORMAT:

LQ\*

### Sample Listing:

#### QUICK BREAKPOINTS

```
1 LOC = ABCD INST = 0F03 (GO)
3                               (DP $P;AR;GO)
```

Directive lines are defined for quick breakpoints 1 and 3, although breakpoint 3 is not currently set. Quick breakpoints 0 and 2 through 9 have neither a defined breakpoint nor a directive line.

### NOTE

Ten quick breakpoints (0 through 9) may be set.  
(See "Set Quick Breakpoint (SQn)".)



## LIST ALL TRUE BREAKPOINTS

### LIST ALL TRUE BREAKPOINTS (L\*)

The List All True Breakpoints (L\*) directive lists all currently defined true breakpoints, their location in memory, the instruction which was replaced, and their associated directive lines. If the work file is not open, the list consists of the locations of the defined true breakpoints and the instruction being replaced. If the work file is open, all defined true breakpoints and all associated directive lines are listed. It is possible to list a true breakpoint without an associated directive line, or a directive line without an associated true breakpoint. However, if neither a true breakpoint nor a directive line is defined for a particular true breakpoint number, that breakpoint number does not appear in the list.

#### FORMAT:

L\*

#### Sample Listing:

```
TRUE BREAKPOINTS
1 LOC = 00ABCD INST = 0F02 ( )
3                               (AR;DP $P;GO)
```

True breakpoint 1 is listed with no directive line and true breakpoint 3 has only a defined directive line. True breakpoints 0, 2, and 4 through 31 have neither a defined true breakpoint nor directive line.

#### NOTE

32 true breakpoints (0 through 31) may be set.  
(See "Set True Breakpoint (Sn)".)

# LIST BOUND UNIT BREAKPOINT

## LIST BOUND UNIT BREAKPOINT (LBn)

The List Bound Unit Breakpoint (LBn) directive displays the stored directive line associated with a specified bound unit breakpoint.

### FORMAT:

LBn

### ARGUMENT:

n

Number of the bound unit breakpoint for which the directive line is to be listed; must be from 0 through 9.

### Example:

LB3

List the directive line associated with bound unit breakpoint 3.

# LIST QUICK BREAKPOINT

## LIST QUICK BREAKPOINT (LQn)

The List Quick Breakpoint (LQn) directive displays a particular quick breakpoint number set by a Set Quick Breakpoint (SQn) directive, and its associated directive line. You can print a directive line without an associated quick breakpoint (e.g., if the quick breakpoint had been previously cleared).

### FORMAT:

LQn

### ARGUMENT:

n

Number of the quick breakpoint whose directive line is listed; must be a decimal digit from 0 through 9.

### Example:

LQ2

Display the directive line associated with quick breakpoint 2.

# LIST TRUE BREAKPOINT

## LIST TRUE BREAKPOINT (Ln)

The List True Breakpoint (Ln) directive displays a particular true breakpoint number set by a Set True Breakpoint (Sn) directive, and its associated directive line.

### FORMAT:

Ln

### ARGUMENT:

n

Number of true breakpoint whose directive line is listed; can be 0 through 31 (decimal).

### Example:

L2

Display the directive line of true breakpoint 2.

## MODE

Change the current mode of the debugger from numeric to symbolic.

### FORMAT:

```
MODE SYM[BOLIC]
```

### DESCRIPTION:

This directive is used when debugging a program written in a high level language, then compiled and linked with the -SYMBOL argument. After invoking the debugger in symbolic mode, you can change to numeric mode by entering:

```
MODE NUM
```

(You might make this change to examine register contents.) Once in numeric mode, you can return to symbolic by entering:

```
MODE SYM
```

### Example:

```
MODE SYM
```

The debugger is currently in numeric mode; the directive changes the mode to symbolic.

### NOTE

A detailed description of the symbolic mode directives and their use is found in Section 7 of this manual.

If the debugger is not initially invoked in symbolic mode, you cannot change from numeric mode to symbolic mode.

# PRINT

## PRINT (Pn)

The Print (Pn) directive prints specified lines predefined by Dn directives. Use the Print All (P\*) directive to print all predefined lines.

### FORMAT:

Pn

### ARGUMENT:

n

Number of the line to be printed; can be 0 through 9.

# PRINT ALL

## PRINT ALL (P\*)

The Print All (P\*) directive prints all lines predefined by Dn directives. Use the Print (Pn) directive to print only specified predefined lines.

### FORMAT:

P\*

# PRINT HEADER LINE

## PRINT HEADER LINE (Hn)

The Print Header Line (Hn) directive prints a specified header line starting at the head of form or after a specified number of lines are skipped. The main uses of the Print Header Line directive are to document printed information related to breakpoint or trace trap debugging, and to annotate a line printer memory dump.

### FORMAT:

Hn (header )

### ARGUMENTS:

n

Number of lines skipped before header line is printed; can be 1 through 9, or 0. 0 causes header to be printed at head of form.

(header )

Any ASCII characters and/or expressions; each expression must be preceded by a percent (%) sign. If a percent sign is to be printed, two percent signs must be used (%%). Left and right parentheses must be balanced within header lines.

### Example:

H0 (DUMP OF BREAKPOINT FOR LEVEL %SS )

Document dumps. As soon as a carriage return is typed, the above header is printed at the top of a new page.



# PRINT HEXADECIMAL VALUE

## PRINT HEXADECIMAL VALUE (VH)

The Print Hexadecimal Value (VH) directive prints, in hexadecimal, the value of each specified expression.

### FORMAT:

VH exp {exp}

### ARGUMENT:

exp

Expression whose value is displayed.

### Example:

VH .+100-M

Display the result of the computation defined by the last referenced memory location plus 100 (hexadecimal) minus the value assigned to the temporary symbol M.

# PRINT QUICK MEMORY POINTER

## PRINT QUICK MEMORY POINTER (PQ)

The Print Quick Memory Pointer (PQ) directive prints the hexadecimal address of the start of quick memory.

FORMAT:

PQ

## PRINT TRACE

### PRINT TRACE (PT)

The Print Trace (PT) directive prints a defined trace directive line.

FORMAT:

PT

# QUIT

## QUIT (QT)

The Quit (QT) directive clears all breakpoints, closes all Debugger work files, and disables the Debugger trap handler before aborting the Multiuser Debugger task.

FORMAT:

QT

# RESET FILE

## RESET FILE (RF)

The Reset File (RF) directive closes files and prohibits execution of directives that refer to user-defined files.

### FORMAT:

RF    { DB }  
      { QK }

### ARGUMENTS:

{ DB }  
{ QK }

DB - Close the Debugger work file and prohibit execution of the P\*, Pn, PT, Sn, Dn, DT, En, SBn, Ln, and LBN directives. These directives may not be entered until another Specify File (SF) directive is issued to open a new work file.

QK - Close the quick disk file and prohibit the quick memory buffers from being written to the file. If no quick breakpoints are currently set prior to an RF directive, the following occurs:

1. The current buffer is marked "last" used and full.
2. The quick disk file is closed after the "last" buffer is written to the disk file.
3. The writer task is terminated.

If there are quick breakpoints still set, steps 1 and 3 are done, but step 2 cannot be guaranteed to write all the buffers to the disk before the file is closed.

# RETURN QUICK MEMORY

## RETURN QUICK MEMORY (RQ)

The Return Quick Memory (RQ) directive causes (1) the quick disk file to be closed after all memory buffers used have been written to it, (2) the asynchronous writer task to be terminated, and (3) memory to be returned to your pool.

### NOTE

Both the quick memory and that memory necessary for quick breakpoint processing are returned to your pool.

### FORMAT:

RQ

# SET BOUND UNIT BREAKPOINT

## SET BOUND UNIT BREAKPOINT (SBn)

The Set Bound Unit Breakpoint (SBn) directive sets a numbered breakpoint for a specified bound unit or overlay. A given bound unit (BU) breakpoint refers to either roots or to overlays, or to both. When the bound unit breakpoint is encountered, a message informs you where the specified bound unit or overlay has been loaded into memory, so that you can then set true breakpoints at specified locations in the program. Because a bound unit is loaded at the time the task associated with it is created, the level number displayed when a BU breakpoint occurs is not necessarily the one used when requests for that task are later executed.

The message format is:

```
*BU n $SL=00xx $E=00xxxx + 00xx
```

n

Number of bound unit breakpoint; can be 0 through 9.

```
$SL=00xx
```

Specifies priority level.

```
$E=00xxxx + 00xx
```

Represents the bound unit base address plus entry point offset as defined by the bound unit or by the caller. Used in place of \$P associated with true breakpoints.

FORMAT:

```
SBn { bound-unit-name  
      bound-unit-name/overlay-number  
      bound-unit-name/*  
      *  
      */overlay-number  
      */* } (directive line)
```

ARGUMENTS:

n

Bound unit breakpoint number; can be from 0 to 9.

## SET BOUND UNIT BREAKPOINT

### bound-unit-name

Name of the bound unit to which the breakpoint applies; up to six ASCII characters (first six characters of the bound unit name).

### overlay-number

Hexadecimal number of the bound unit overlay.

\*

Stands for all roots or all overlays, depending on context.

### (directive line)

Directives to be executed when the bound unit or overlay is loaded.

### Example:

```
SB6 S00Z/A (IF 3D02=5354;VH M-2;GO)
```

Sets breakpoint 6 for overlay number A (hexadecimal) of the bound unit named S00Z. The directive line specifies that if the condition indicated is true (location 3D02 equals 5354), then the value of M minus 2 is displayed. When overlay A is loaded into memory, its location is displayed at the terminal, and the directive line associated with bound unit breakpoint 6 is executed.



## SET EXPRESS BOUND UNIT BREAKPOINT

### SET EXPRESS BOUND UNIT BREAKPOINT (XBn)

The Set Express Bound Unit Breakpoint (XBn) directive sets a numbered breakpoint for a specified bound unit or overlay. A given bound unit (BU) breakpoint refers to either roots or to overlays, or to both. This directive operates similarly to the Set Bound Unit Breakpoint (SBn) directive, however instead of stopping when the bound unit breakpoint is encountered, a true breakpoint is set and execution continues until the true breakpoint is encountered.

Express bound unit breakpoints are not another set of bound unit breakpoints, but are simply a mode of bound unit breakpoints. Therefore, all of the features of normal bound unit breakpoints are available in express mode. This includes the ability to include a stored directive with the express bound unit by opening an xxx.DB file with the Specify File (SF) directive.

The breakpoint number for the true breakpoint which is set when the express bound unit breakpoint is encountered is the same as the number of the express bound unit breakpoint itself. This true breakpoint must be in the cleared state when the express bound unit breakpoint is set. Otherwise, a Breakpoint Active error will occur. Conversely, setting a true breakpoint when the express bound unit breakpoint of the same number is set will also result in a Breakpoint Active error.

An optional argument can be given with the express mode directive to specify the offset into the bound unit at which the true breakpoint will be set. If no offset is given or if the given offset is zero, the true breakpoint is set at the default entry point (\$E).

When the true breakpoint resulting from an express breakpoint is encountered, an internally generated directive is issued to clear the true breakpoint. This frees the true breakpoint for re-use by the express bound unit breakpoint. For this reason, if such a true breakpoint has a stored directive associated with it, the stored directive is not executed.

#### FORMAT:

```
XBn { bound-unit-name
      bound-unit-name/overlay-number
      bound-unit-name/*
      *
      */overlay-number
      */* } [offset [(directive line)]]
```

## SET EXPRESS BOUND UNIT BREAKPOINT

### ARGUMENTS:

n

Bound unit breakpoint number; can be from 0 to 9.

bound-unit-name

Name of the bound unit to which the breakpoint applies; up to twelve ASCII characters (first twelve characters of the bound unit name).

overlay-number

Hexadecimal number of the bound unit overlay as given in the link map.

\*

Stands for all roots or all overlays, depending on context.

offset

Relative offset from the beginning of the load unit at which to set the true breakpoint. An offset must be specified if a directive line is to be used. The offset is limited to four hexadecimal digits.

directive line

Directives to be executed when the bound unit or overlay is loaded. When an express bound unit breakpoint has a non-null directive associated with it, that directive will be executed in lieu of the internally generated directive which sets the corresponding true breakpoint. If this function is required, include a Set True Breakpoint (Sn) directive. A GO directive is not required, since a GO is always issued in express mode.

When stored directives are used with express bound unit breakpoints, it is possible to include a set true breakpoint directive with a number differing from that of the express bound unit breakpoint. If this is done, the true breakpoint is not cleared when encountered and any directives associated with it are executed.

## SET EXPRESS BOUND UNIT BREAKPOINT

Example:

XB6 S00Z/A

Sets breakpoint 6 for overlay number A (hexadecimal) of the bound unit named S00Z. When overlay A is loaded into memory, true breakpoint 6 is set and execution continues until the true breakpoint is encountered. When the true breakpoint is encountered, an internally generated directive is issued to clear the true breakpoint. This frees the true breakpoint for re-use by the express bound unit breakpoint. If the true breakpoint has a stored directive associated with it, the stored directive is not executed.

# SET QUICK BREAKPOINT

## SET QUICK BREAKPOINT (SQn)

The Set Quick Breakpoint (SQn) directive sets a numbered quick breakpoint at a specified location. When the breakpoint is encountered, the stored specified directive line is executed and the Debugger task continues. A message and any information requested by the directive line are written to quick memory and, optionally, to a quick disk file. The entire SQn directive may comprise a maximum of 124 characters.

If there is a preexisting directive line associated with a given quick breakpoint and that directive line is no longer applicable, clear the line by designating empty parentheses ( ) when resetting the quick breakpoint.

The message format is:

(\$\$)QBn Group Id TCB ptr Level

n

Quick breakpoint number; must be 0 through 9.

Group Id

Group name of the task being debugged.

TCB ptr

Task control block location of the task being debugged.

Level

Priority level of the task being debugged.

### NOTES

1. A quick breakpoint cannot be set in any of the following instructions: input/output, generic (BRK), scientific, invalid instruction, LEV, ENT, LNJ, JMP, STS, or any instruction with an invalid address symbol.
2. A GO directive should be the last directive specified in a quick breakpoint directive line. A GO directive embedded in an SQn directive allows task execution to proceed after the desired operation has been performed. The Multiuser Debugger appends a GO directive to the directive line.

## SET QUICK BREAKPOINT

3. If the NR argument is specified, there is no evidence that the quick breakpoint has been encountered. If the directive line contains an IF directive and the condition specified is true, any requested information is stored in the memory buffer.

### FORMAT:

SQLn exp {NR} (directive line)

### ARGUMENTS:

n

Number of the quick breakpoint; can be 0 through 9.

exp

Location at which the quick breakpoint occurs.

{NR}

Quick breakpoint output is not stored in the memory buffer.

(directive line)

Directives that are executed when the quick breakpoint is reached. The directives allowed in a quick breakpoint directive line are: AR, AS, CH, DH, GO, HS, IF, and VH. The GO directive should only appear as the last entry in the directive line; if omitted, the GO directive is appended.

If GO is the only directive specified in the directive line, only the message described above is stored in the memory buffer.

### Example:

```
SQL1 1D8B NR (IF 1000<,3E;AR;GO)
```

A quick breakpoint numbered "1" is set at location 1D8B. If the condition specified in the directive line is false, no information is stored in the memory buffer. If the condition is true, the breakpoint message and the contents of the active registers are stored in the memory buffer.

**SET TEMPORARY LEVEL**

(DELETED)

# SET TRUE BREAKPOINT

## SET TRUE BREAKPOINT (Sn)

The Set True Breakpoint (Sn) directive sets a numbered true breakpoint at a specified location. When the true breakpoint is encountered, the stored specified directive line, if any, is executed. Otherwise, a typeout occurs indicating the contents of the location counter and the active priority level, and the task execution is suspended. The Set File (SF) directive is a precondition for directive line execution. The entire Sn directive may comprise a maximum of 126 characters.

If there is a preexisting directive line associated with a given true breakpoint and that directive line is no longer applicable, clear the line by designating empty parentheses ( ) when setting the true breakpoint.

The message format is:

(\$H) BPn \$P=00xxxx \$SL=00xx

\$P=00xxxx

Location counter

\$SL=00xx

Priority level

### NOTES

1. If a true breakpoint is set in any of the following types of instructions, the true breakpoint must be cleared (Cn directive) before continuing execution (GO directive): input/output, generic (BRK), scientific, LEV, invalid instruction, or instruction with an invalid address syllable. To avoid this restriction, clear the existing true breakpoint and then reset it in a subsequent Sn directive.
2. A GO directive embedded in an Sn directive line allows task execution to proceed after the desired operations have been performed, without further operator intervention.
3. True breakpoints are cleared when a bound unit or overlay is loaded or unloaded. Therefore, you should not clear previous breakpoints when loading a new bound unit.

## SET TRUE BREAKPOINT

### FORMAT:

Sn exp {(directive line)}

### ARGUMENTS:

n

Number of true breakpoint; can be 0 through 31 (decimal).

exp

Location at which true breakpoint occurs:

{(directive line)}

Directives that are executed when true breakpoint is encountered.

### Example 1:

```
S0 100 (DH 200/10;GO)
```

Display locations 200 to 20F when location 100 is reached, then proceed from breakpoint.

### Example 2:

```
S0 100 ( )
```

Cancel any line previously associated with true breakpoint 0.

### Example 3:

```
S0 1000 (AR;C0;GO)
S1 1003 (S0 1000;GO)
```

The first directive line sets true breakpoint number 0 at location 1000, prints all registers on the active level, clears true breakpoint number 0 because the instruction at location 1000 is restricted (see Note 1 above), then proceeds from the breakpoint.

The second directive line sets true breakpoint number 1 at location 1003 and then reestablishes true breakpoint 0 at location 1000; the second true breakpoint line causes no visible action except the printing of the breakpoint message.



# SLEEP

## SLEEP (SP)

The Sleep (SP) directive temporarily suspends the execution of the Multiuser Debugger and returns control to the command processor.

### FORMAT:

SP

# SPECIFY FILE

## SPECIFY FILE (SF)

The Specify File (SF) directive identifies the relative or full pathname of the Multiuser Debugger file to be opened. Since the function of the SF directive is to locate the file, first execute this directive; otherwise, an error message appears as soon as a directive requiring the file is used. When using quick breakpoints, the first directive entered after invoking the Multiuser Debugger should be the Get Quick Memory directive. The Specify File directive, in this case, should be the second directive entered.

### FORMAT:

SF { path  
path -CYCLE }  
-CYCLE }

### ARGUMENTS:

path

Relative or full pathname of the file to be opened; relative pathname can be 1 to 12 characters in length. All Multiuser Debugger work files must end with the suffix .DB; all Multiuser Debugger quick disk files must end with the suffix .QK.

### -CYCLE

Used only with quick disk files. At end of file, returns the Debugger writer task, which enters debug information into the file, to the beginning of the quick disk file. If -CYCLE is not specified, the quick disk file is closed at end of file even if there was more data to be written.

### NOTE

If you did not initially specify the -CYCLE argument and desire to do so later in the Debugger session, enter

SF -CYCLE

at any point later in the Debugger directive sequence before end of file; this causes the Debugger writer task to return to the beginning of the quick disk file when it reaches end of file.

Example 1:

```
SF GLASS.DB
```

Work file GLASS.DB is opened.

Example 2:

```
SF GLASS.QK -CYCLE
```

Quick disk file GLASS.QK is opened and, when end of file is reached, the Debugger writer task returns to the beginning of the quick disk file to continue entering input into the file.

Example 3:

```
SF GLASS.QK
```

```
.
```

```
.
```

```
.
```

```
SF -CYCLE
```

Quick disk file GLASS.QK is opened. Later in the program, force the writer task to return to the beginning of the quick disk file to complete the writing task.

#### NOTES

1. If the .QK or .DB suffix is not specified in the SF directive line for a work file, it is assumed a work file is being requested and .DB is appended before the file is opened.
2. If the specified work file does not exist, it is created and opened with exclusive read/write access when the SF directive is entered. Only one user has access to a Debugger work file at any given time.
3. If a simple pathname is entered, the system looks only in the current working directory for the specified work file.
4. You have the option of changing work files by entering a new SF directive, thereby closing the currently active file and opening a new file in its place.

## SPECIFY FILE

5. The quick disk file must have been previously created outside the Multiuser Debugger task using the Create File (CR) command. The path-name supplied must include the suffix .QK. The argument values supplied must agree with those used in the Get Quick Memory (MQ) directive. In the following example, the argument values are the default values of the MQ directive:

```
CR filename.QK -REL -CISZ 4096 -SZ n -LRSZ 512
```

n must be greater than or equal to 2.

6. The .QK suffix must be specified in the SF directive line for a quick disk file; otherwise, a default value of .DB is appended and a work file is opened.
7. The quick disk file can only be opened when running the Multiuser Debugger from the system (\$S) group. Only one quick disk file can be opened at any given time. When the SF directive is specified, this quick disk file has exclusive write access.
8. Opening a quick disk file spawns the Debugger writer task. The writer task terminates when the quick disk file is closed. This task runs on level 62.
9. You have the option of changing the quick disk file by entering a Reset File (RF) directive, thereby closing the currently active file. Then enter a new Specify File (SF) directive to open the new file.

## START J-MODE TRACE

### START J-MODE TRACE (ST)

The Start J-Mode Trace (ST) directive sets the given task's M1 register j-bit on. As a result, any departure from the current processing sequence causes a trap. The Multiuser Debugger treats the trap as a "trace trap." The following points apply:

- j-mode trace can be started only for a task which is currently suspended due to a true breakpoint.
- The ST directive is refused if the task is suspended due to a bound unit breakpoint.
- j-mode processing is specific to a given task and is shut off or restored at the monitor call interfaces.
- When a task is running in j-mode, the Multiuser Debugger's handling of successive traps is governed by the limit-to-pause counter of the GO directive.
- Limit-to-pause has a default value of 1, but may be set to an arbitrary value via the GO directive. The Multiuser Debugger decrements the limit-to-pause once for each occurrence of a trace trap. When limit-to-pause assumes the value zero, the trapped task is suspended to permit operator action and a TRACE PAUSE message is issued. When the task is reactivated (GO [ LLLL]) the limit-to-pause is reset to the default value or to a user-specified value.

#### FORMAT:

ST lvl

#### ARGUMENT:

lvl

Active level of the task in question.

## **TURN ON ABNORMAL TRAP BIT**

### TURN ON ABNORMAL TRAP BIT (TB)

Turn on the abnormal trap bit in the debugger's indicator word.

This bit is set to request that a special breakpoint message be displayed if a task in the specified group encounters an unexpected (abnormal) 0303xx trap condition. With the bit set, the trap information is displayed, the task is suspended, and the special breakpoint message is displayed. At this time, debug has control of the group, allowing the user to determine what caused the trap. The user can then decide to continue from the trap (by entering GO) or to terminate the task (by entering TT).

This bit is automatically set when debug is first invoked in a group. It can be turned off at any time by entering the Clear Abnormal Trap Bit (CT) directive.

The format of the abnormal breakpoint message is:

BP TP \$SSL=00XX \$P=00XXXX

\$P points to the next location in memory following the trapped instruction.

**FORMAT:**

**TB**

## TERMINATE THE TRAPPED TASK

### TERMINATE THE TRAPPED TASK (TT)

Terminate the request previously entered against the trapped task.

A user who decides that the task being debugged has been sufficiently examined can terminate the task by this directive. The TT directive can be executed only for a task suspended by a true or special trap breakpoint.

#### NOTE

If TT terminates a task that has been abnormally trapped and is now suspended on the special breakpoint, there is no evidence of that task left in the task group. Normally, the trapped task's TCB and associated TSAs are left in the group for later analysis of a dump.

#### FORMAT:

TT

## MULTIUSER DEBUGGER (NUMERIC MODE) PROCEDURES

Three sample debugging sessions are shown below to illustrate some of the directives and procedures described earlier in this section. The second and third debugging sessions illustrate primarily the use of quick breakpoints.

### Sample Session 1

The bound unit being debugged is TEST, listed in Figure 8-1. TEST takes as an argument a number in the range 0 through 2. The function of TEST is to write to user-out one of three numbered messages; the message number should correspond to the number entered as the argument.

The debugging session is shown in Figure 8-2. At the start of the listing shown in Figure 8-2, TEST is invoked with the argument 1. TEST should write to user-out TEST MESSAGE ONE, but fails to do so. A debugging session follows. Each Debugger directive beside which a number appears is explained by a correspondingly-numbered comment, beginning below.

1. Invoke the Multiuser Debugger.
2. Set bound unit breakpoint 1 on bound unit TEST.
3. List all bound unit breakpoints.
4. Put the debugger to sleep (SP) to allow input to the group through ECL commands. Note that the group is back in RDY state, waiting for input.
5. Type the bound unit name with an argument, causing a bound unit breakpoint message to appear. You are now back in debug mode, ready to type in debug directives.
6. Assign temporary symbol X to the base of the program in memory. If the start address of the program is not offset zero, the offset value must be subtracted from \$E to determine the base.

Example:

Assume the following bound unit breakpoint message:

```
*BU 2  $SL=001B  $E=00ABCD + 0023  DATA=000000
```

To set X to the base of the bound unit, you would type

```
AS X $E-23
```

or

```
AS X ABCD
```



```

TITLE TEST          07/22/80  1557.8  edt  Tue          HRS ASSEMBLER 6.02  -SLIC PAGE 0001
000001              TITLE          test,'80072200'      Debug Test Program
000002              *
000003              *          COPYRIGHT, (C), 1980, HONEYWELL INFORMATION SYSTEMS, INC.
000004              *
000005              0000          *
000006  0000  F877          start    equ    $
000007  0001  9870 1702      ldr    $r7,+ $b7          arg count from IAB
000008  0003  7002          ldr    $r1,=x'1702'      error status
000009  0004  0216          cmv    $r7,=2          two or more args?
000010  0005  80F7          bl    >error          br if no
000011  0006  9C87          cmn    + $b7          move passed first arg
000012  0007  9871          ldb    $b1,$b7          ptr to second argument
000013  0008  8AD1          ldr    $r1,+ $b1          arg size
000014  0009  1001          inc    = $r1
000015              sol    $r1,1          convert to words
000016              *
000017  000A  A0*1          arg    equ    $
000018  000B  2E00          ldh    $r2,$b1          get second arg
000019  000C  ABC0 0017      adv    $r2,-x'30'      drop ascii bits
000020  000E  E822          lab    $b2,msgdsp
000021  000F  A0D6          ldr    $r6,$b2,$r2          length of arg
000022  0010  A570 00FF      ldh    $r2,= $r6          get length
000023  0012  C0C0 0015      and    $r2,=x'00FF'      drop sign extension
000024  0014  CCA4          lab    $b4,msg          ptr to msg table
000025  0015  6048          ldb    $b4,$b4,$r2          move to desired msg
000026  0016  7C00          sor    $r6,8          shift length to right byte
000027  0017  0001          ldv    $r7,0
000028  0018  0801          mcl
000029  0019  1907          dc    x'0801'          call USER_OUT
000030  001A  B870 0080      bez    $r1,>done          br if no error
000031  001C  F851          error  ldr    $r3,=x'80'      component code
000032  001D  6C00          ldr    $r7,= $r1          error code
000033  001E  0001          ldv    $r6,0
000034  001F  0F00          mcl
000035              dc    x'0F00'          call REPORT ERROR
000036              *
000037              0020          done    equ    $
000038  0020  1C00          ldv    $r1,0
000039  0021  2C00          ldv    $r2,0
000040  0022  0001          mcl
000041  0023  0103          ic    x'0103'          call TERMINATE TASK
000042              *
000043  0024  0604          msgdsp equ    $
000044  0025  0E90          dc    msg0-msgdsp+256*(msg0_l-1)
000045  0026  0614          dc    msg1-msgdsp+256*(msg1_l-1)
                                dc    msg2-msgdsp+256*(msg2_l-1)

```

Figure 8-1. Sample Program TEST

8-72

CZ15-02

```

                                07/22/80 1557.8 edt Tue           HRS ASSEMBLER 6.02 -SLIC PAGE 0002
000046 0027 0910                                dc           msg3-msgdsp+256*(msg3_l-1)
000047                                0028                                eq           $
000048 0028 4140 4553 5341                                msg0         dc           'MESSAGE ZERO '
                                4745 205A 4552
                                4F20
000049                                0007                                msg0_l       equ           $-msg0
000050 002F 4154 4553 5420                                msg1         dc           'ATEST MESSAGE ONE '
                                4045 5353 4147
                                4520 4F4E 4520
000051                                0009                                msg1_l       equ           $-msg1           message length
000052 0038 4140 4553 5341                                msg2         dc           'MESSAGE TWO '
                                4745 2054 574F
                                2020
000053                                0007                                msg2_l       equ           $-msg2
000054 003F 4154 4553 5420                                msg3         dc           'ATEST MESSAGE THREE '
                                4045 5353 4147
                                4520 5448 5245
                                4520
000055                                000A                                msg3_l       equ           $-msg3
000056                                **
000057 0049 0000                                end           test,start
0000 ERR COUNT
```

Figure 8-1 (cont). Sample Program TEST

```

($H)RDY:
TEST 1
($H),/D
($H)RDY:
1  DEBUG
  ($H)DEBUG-R210-07/18/1310
2  SB1 TEST
3  LB*
  ($H) BU1 TEST
4  SP
  ($H)RDY:
5  TEST 1
  ($H) *BU 1 $SL=001B $E=00FDDA + 0000 DATA+000000
6  AS X $E
7  VH X
  ($H) X=00FDDA
8  DP X/10
  ($H)
  ($H) 00FDDA/ F877 9870 1702 7D02 0216 8DE7 9C87 9871 .w.p..}.....8
  ($H) 00FDE2/ BAD1 1001 A081 2ED0 ABC0 0017 E822 A0D6 .....".
9  S1 X
10 L*
  ($H) TRUE BREAKPOINTS
  ($H) 1 LOC=00FDDA INST=F877
11 DP X
  ($H)
  ($H) 00FDA/ 0002 9870 1702 7D02 0216 8DF7 9C87 9871 ...p..}.....q
12 GO
  ($H) *BP 1 $SL=001C $P=00FDDA
13 AR
  ($H) $R1=0000 $R2=0000 $R3=0000 $R4=0000 $R5=0000 $R6=0000
  ($H) $R7=0000 $B1=000000 $B2=00FFA6 $B3=000000 $B4=00FFA2
  ($H) $B5=001194 $B6=00FDDA $B7=00FFA6 $P=00FDDA $I=0000 $S=4010
14 DP $B7/10
  ($H)
  ($H) 00FFA6/ 0002 FFAA FFAE 0000 0004 5445 5354 2020 .....TEST
  ($H) 00FFAE/ 0001 3120 0102 FFA2 FD82 FD42 0000 8002 ..1.....B....
15 S2 X+A
16 GO
  ($H) *BP 2 $SL=001C $P=00FDE4
17 AR
  ($H) $R1=0004 $R2=0000 $R3=0000 $R4=0000 $R5=0000 $R6=0000
  ($H) $R7=0002 $B1=00FFAF $B2=00FFA6 $B3=000000 $B4=00FFA2
  ($H) $B5=001194 $B6=00FDDA $B7=00FFA8 $P=00FDE4 $I=0004 $S=401C
18 AS $R1 1
19 DP $B1
  ($H)
  ($H) 00FFAF/ 3120 0102 FFA2 FD82 0000 8002 0000 1 .....B.....
20 DP X/10
  ($H)
  ($H) 00FDDA/ 0002 9870 1702 7D02 0216 8DF7 9087 9871 ...p..}.....q
  ($H) 00FFE2/ 8AD1 1001 0002 2ED0 ABC0 0017 E822 A0D6 .....".
21 S3 X+B
22 L*
  ($H) TRUE BREAKPOINTS
  ($H) 1 LOC=00FDDA INST=F877
  ($H) 2 LOC=00FDE4 INST=A081
  ($H) 3 LOC=00FD35 INST=2ED0

```

Figure 8-2. Debugging Session of TEST

```

23 GO
  ($H) *BP 3 $SL=001C $P=00FDE5
24 AR
  ($H) $R1=0001 $R2=0031 $R3=0000 $R4=0000 $R5=0000 $R6=0000
  ($H) $R7=0002 $B1=00FFAF $B2=00FFA6 $B3=000000 $B4=00FFA2
  ($H) $B5=001194 $B6=00FDDA $B7=00FFA8 $P=00FDE5 $I=0004 $S=401C
25 S4 X+F
26 GO
  ($H) *BP 4 $SL=001C $P=00FDE9
27 AR
  ($H) $R1=0001 $R2=0001 $R3=0000 $R4=0000 $R5=0000 $R6=080B
  ($H) $R7=0002 $B1=00FFAF $B2=00FDFF $B3=000000 $B4=00FFA2
  ($H) $B5=001194 $B6=00FDDA $B7=00FFA8 $P=00FDE9 $I=0024 $S=401C
28 DP $B2
  ($H)
  ($H) 00FDFF/ 0604 080B 0614 091B 4140 4553 5341 4745 .....AMESSAGE
29 S5 X+14
30 GO
  ($H) *B 5 $SL=001C $P=00FDEE
31 AR
  ($H) $R1=0001 $R2=000B $R3=0000 $R4=0000 $R5=0000 $R6=080B
  ($H) $R7=0002 $B1=00FFAF $B2=00FDFF $B3=000000 $B4=00FE02
  ($H) $B5=001194 $B6=00FDDA $B7=00FFA8 $P=00FDEE $I=0024 $S=401C
32 DP $B4
  ($H)
  ($H) 00FE02/ 414D 4553 5341 4745 205A 4552 4F20 4154 AMESSAGE ZERO AT
33 DP $B4/10
  ($H)
  ($H) 00FE02/ 414D 4553 5341 4745 205A 4552 4F20 4154 AMESSAGE ZERO AT
  ($H) 00FE0A/ 4553 5420 4D45 5353 4147 4520 4F4E 4520 EST MESSAGE ONE
34 S6 X+17
35 L*
  ($H) TRUE BREAKPOINTS
  ($H) 1 LOC=00FDDA INST=F877
  ($H) 2 LOC=00FDE4 INST=A081
  ($H) 3 LOC 00FDE5 INST=2ED0
  ($H) 4 LOC 00FDE9 INST=A0D6
  ($H) 5 LOC=00FDEE INST=00A4
  ($H) 6 LOC=00FDF1 INST=0001
36 GO
  ($H) *BP 6 $SL=001C $P=00FDF1
37 AR
  ($H) $R1=0001 $R2=000B $R3=0000 $R4=0000 $R5=0000 $R6=0008
  ($H) $R7=0000 $B1=00FFAF $B2=00FDFF $B3=000000 $B4=005353
  ($H) $B5=001194 $B6=00FDDA $B7=00FFA8 $P=00FDF1 $I=0004 $S=4010
38 DP $B4
  ($H)
  ($H) 005353/ 83C8 FF68 190E A870 0008 F851 E870 0000 ...h...p...Q.p..
39 DH $B2+B
  ($H) 00FE09/ 4154
40 DP FE09
  ($H)
  ($H) 00FE09/ 4154 4553 5420 4D45 5353 4147 4520 4F4E ATEST MESSAGE ON
41 AS B4 FE09
42 DP $B4
  ($H)
  ($H) 00FE09/ 4154 4553 5420 4D45 5353 4147 4520 4F4E ATEST MESSAGE ON
43 DP $B4/10
  ($H)
  ($H) 00FE09/ 4154 4553 5420 4D45 5353 4147 4520 4F4E ATEST MESSAGE ON
  ($H) 00FE11/ 4520 414D 4553 5341 4745 2054 574F 2020 E AMESSAGE TWO

```

Figure 8-2 (cont). Debugging Session of TEST

```

44 GO
  ($H) ILL INST "GO"
45 C6
46 GO
  ($H) TEST ME
  ($H) RDY:
47 TEST 1
  ($H) *BU 1 $SL=001B $E=00FDDA + 0000 DATA=000000
48 DP $E/20
  ($H)
  ($H) 00FDDA/ F877 9870 1702 7D02 0216 8DF7 9087 9871 .w.p..}.....q
  ($H) 00FDE2/ 8AD1 1001 A081 2ED0 ABC0 0017 E822 A0D6 ....."}..
  ($H) 00FDEA/ A570 00FF CBC0 0015 CCA4 6048 7000 0001 .p.....'H ...
  ($H) 00FDF2/ 0801 1907 B870 0080 F851 6C00 0001 0F00 .....p...Q .....
49 L*
  ($H) TRUE BREAKPOINTS
  ($H) 1 LOC=00FDDA INST=F877
  ($H) 2 LOC=00FDE4 INST=A081
  ($H) 3 LOC=00FD35 INST=2ED0
  ($H) 4 LOC=00FDE9 INST=A0D6
  ($H) 5 LOC=00FDEE INST=CCA4
50 C*
51 L*
  ($H) INACTIVE BP "L*"
52 DP $E/20
  ($H)
  ($H) 00FDDA/ F877 9870 1702 7D02 0216 8DF7 9087 9871 .w.p..}.....q
  ($H) 00FDE2/ 8AD1 1001 A081 2ED0 ABC0 0017 E822 A0D6 ....."}..
  ($H) 00FDEA/ A570 00FF CBC0 0015 CCA4 6048 7000 0001 .p.....'H ...
  ($H) 00FDF2/ 0801 1907 B870 0080 F851 6C00 0001 0F00 .....p...Q .....
53 S10 $E
54 GO
  ($H) *BP 10 $SL=001C $P=00FDDA
55 DP $P
  ($H)
  ($H) 00FDDA/ 0002 9870 1702 7D02 0216 8DF7 9087 9871 ...p..}.....q
56 AS X $P
57 VH X
  ($H) X=00FDDA
58 CH X+14 CBA2
59 DP X+14
  ($H)
  ($H) 00FDEF/ CBA2 6048 7000 0001 0801 1907 B870 0080 ..'H .....p.
60 DH X+25
  ($H) 00FDF0/ 080B
61 CH X+25 100B
62 DH X+25
  ($H) 00FDF0/ 100B
63 S11 X+16
64 GO
  ($H) *BP 11 $SL=001C $P=00FDF0
65 AR
  ($H) $R1=0004 $R2=000B $R3=0000 $R4=0000 $R5=0000 $R6=0010
  ($H) $R7=0002 $B1=00FFAF $B2=00FDFF $B3=000000 $B4=00FE09
  ($H) $B5=001194 $B6=00FDDA $B7=00FFA8 $P=00FDF0 $I=000F $S=4010
66 DP $B4/10
  ($H)
  ($H) 00FE09/ 4154 4553 5420 4D45 5353 4147 4520 4F4E ATEST MESSAGE ON
  ($H) 00FE11/ 4520 414D 4553 5341 4745 2054 574F 2020 E AMESSAGE TWO
67 GO
  ($H) TEST MESSAGE ON
  ($H) RDY

```

Figure 8-2 (cont). Debugging Session of TEST

7. Verify the value that has been assigned to X.
8. Display memory starting at the location assigned to X for 10 (hexadecimal) locations.
9. Set true breakpoint 1 at location X (the base of the bound unit).
10. List all true breakpoints currently set.
11. Display memory starting at location X. Note that the instruction F877 has been replaced by 0002--a break instruction. The original instruction has been saved in a table in Debugger workspace.
12. Reactivate the broken task by typing GO. The GO directive causes the true breakpoint to be encountered and the message to appear. GO must be typed from a true breakpoint; SP is not accepted at this time.
13. Display all registers. You must be at a true breakpoint for register values to be meaningful. Register values displayed at a bound unit breakpoint are not meaningful for the bound unit being debugged.
14. Display memory pointed to by \$B7 for 10 (hexadecimal) locations.
15. Set true breakpoint 2 at location X + A.
16. Type GO, causing true breakpoint 2 to be encountered and the message to appear.
17. Display all registers.
18. By means of the AS directive, change the value of \$R1 from 4 to 1. The logic of the program calls for a division by 2 to convert the number of bytes to words. Instead, the instruction at offset 9 multiplies by 2. The value of \$R1 is changed to correct this mistake.
19. Display memory pointed to by \$B1.
20. Display memory starting at X, to show where breakpoints are currently set.
21. Set true breakpoint 3.
22. List all currently active breakpoints.
23. GO from breakpoint 2.
24. Display all registers.

25. Set true breakpoint 4.
26. GO from breakpoint 3.
27. Display all registers.
28. Display memory pointed to by \$B2.
29. Set true breakpoint 5.
30. GO from breakpoint 4.
31. Display all registers.
32. Display memory pointed to by \$B4.
33. Display more of memory pointed to by \$B4 than was requested for display by the previous directive.
34. Set true breakpoint 6.
35. List all currently active true breakpoints.
36. GO from breakpoint 5.
37. Display all registers.
38. Display memory pointed to by \$B4.
39. Display in hexadecimal only (not in ASCII) memory at the location pointed to by \$B2 + B.
40. Display memory at location FE09.
41. Assign FE09 to \$B4, which was not pointing to the proper location.
42. Display memory pointed to by \$B4 to confirm that the value just assigned to \$B4 is correct.
43. Display more of memory pointed to by \$B4.
44. GO from breakpoint 6. The message ILL INST "GO" means that the breakpoint must be cleared before the GO can be issued. The description of the Set True Breakpoint directive (earlier in this section) explains when a breakpoint must be cleared before GO can be issued.
45. Clear breakpoint 6, replacing the 0002 break instruction with the original instruction, which has been stored in Debugger workspace.
46. GO from breakpoint 6. There are no more breakpoints, and the bound unit completes execution. The group is back in ECL mode, awaiting input (RDY).

47. Type in the bound unit name with argument in order to step through the program again. Note that the bound unit breakpoint is still set for TEST.
48. Display memory starting at the base address of the bound unit (\$E).
49. List all currently active breakpoints. Even though the Debugger thinks that true breakpoints 1 through 5 are active, there are no 0002 instructions in memory at the specified locations. When TEST is reinvoked, a new copy of the bound unit is loaded in memory, overwriting the version containing the breakpoint instructions. It is important to remember that true breakpoints must be reset after each invocation of a program.
50. Clear all currently active breakpoints.
51. List all currently active breakpoints.
52. Display memory starting at the base address (\$E) of the bound unit. You can enter the expression \$E only when at a bound unit breakpoint. At other times, refer to the value of \$E by assigning that value a temporary symbol in the range G through Z.
53. Set true breakpoint 10. It is possible to reuse breakpoint numbers 1 through 6. The number 10 was chosen simply to show that higher numbers are available.
54. GO from bound unit breakpoint 1.
55. Display memory pointed to by \$P (program counter).
56. Assign the value of \$P to the temporary symbol X.
57. Verify the value assigned to X.
58. Change the instruction at offset 14 from LDB to LAB.
59. Display memory starting a location X + 14 of the bound unit. (The displayed change has, of course, occurred only in memory.)
60. Display the contents of memory at offset 25 of the bound unit.
61. Change the value of the memory location X + 25.
62. Display the location again to view the new contents.
63. Set true breakpoint 11.
64. GO from breakpoint 10.



65. Display all registers.
66. Display memory pointed to by \$B4.
67. GO from breakpoint 11. Because no more breakpoints have been set, the bound unit completes execution; the group is back at the RDY state, awaiting input.

### Sample Session 2

The bound unit TSTNOW, listed in Figure 8-3, is debugged with quick breakpoints. The debugging session is shown in Figure 8-4. Each numbered Debugger directive in Figure 8-4 is explained below by a correspondingly-numbered comment.

1. Establish standard I/O files for the system (\$S) group.
2. Turn on the ready prompt. (Use of the ready prompt is optional. In this example, RDY helps to distinguish user input from system response.)
3. Change the default group id to \$H.
4. Change the working directory of the \$H group.
5. Create a quick disk file, specifying for control interval and logical record size the default values of Get Quick Memory (MQ) arguments.
6. Change the default group id to \$S. To use quick breakpoints, you must invoke the Debugger from the \$S group.
7. Invoke the Multiuser Debugger. (The bound unit usually resides in SYSLIB2 and is invoked by entering DEBUG.)
8. Request quick memory, using the default values.
9. Print the memory location at which quick memory begins.
10. Open the quick disk file.
11. Set bound unit breakpoint one on the bound unit TSTNOW.
12. List all bound unit breakpoints currently set.
13. Put the Debugger to sleep. This directive returns the group (\$S) to the ready state, allowing you to enter ECL commands.
14. Invoke the bound unit TSTNOW. This command causes bound unit breakpoint 1 to be encountered and its breakpoint message to be displayed. The occurrence of breakpoint one reactivates the debugger, which handles all input to the \$S group until GO or QT is entered.

| TITLE  | TSTNOW | 08/13/82 | 1334.4 | edt | Fri   | HRS ASSEMBLER 9.00                                   | -LAF           | PAGE | 0001 |
|--------|--------|----------|--------|-----|-------|--|----------------|------|------|
| 000001 |        |          |        |     | title | tstnow   |                |      |      |
| 000002 |        |          |        |     | *     |  |                |      |      |
| 000003 |        |          |        |     | *     |  |                |      |      |
| 000004 |        |          |        |     | *     | COPYRIGHT, (C), 1981, HONEYWELL INFORMATION SYSTEMS, |                |      |      |
| 000005 | 0000   | CBC0     | 0036   |     | begin | lab  | \$b4,buffer    |      |      |
| 000006 | 0002   | E870     | 401C   |     |       | ldr  | \$r6,=Z'401C'  |      |      |
| 000007 | 0004   | 1C05     |        |     |       | ldv  | \$r1,5         |      |      |
| 000008 |        |          |        |     | *     |  |                |      |      |
| 000009 | 0005   | 0001     |        |     |       | mcl  |                |      |      |
| 000010 | 0006   | 0C08     |        |     |       | dc   | x'0C08'        |      |      |
| 000011 |        |          |        |     | *     |  |                |      |      |
| 000012 | 0007   | 19AA     |        |     |       | bnez   | \$r1,>aa       |      |      |
| 000013 |        |          |        |     | *     |  |                |      |      |
| 000014 |        |          |        |     | *     | \$USOUT  | !SLEW,=29,=1   |      |      |
| 000015 |        |          |        |     | *     | \$LDB  | \$B4,!SLEW     |      |      |
| 000016 | 0008   | CBC0     | 002D   |     |       | lab  | \$b4,slew      |      |      |
| 000017 | 000A   | E870     | 001D   |     |       | ldr  | \$r6,=29       |      |      |
| 000018 | 000C   | F870     | 0001   |     |       | ldr  | \$r7,=1        |      |      |
| 000019 | 000E   | 0001     |        |     |       | mcl  |                |      |      |
| 000020 | 000F   | 0801     |        |     |       | dc   | x'0801'        |      |      |
| 000021 |        |          |        |     | *     |  |                |      |      |
| 000022 |        | 0010     |        |     | user  | equ  | \$             |      |      |
| 000023 | 0010   | CBC0     | 003A   |     |       | lab  | \$b4,buff_2    |      |      |
| 000024 | 0012   | E870     | 401E   |     |       | ldr  | \$r6,=Z'401E'  |      |      |
| 000025 | 0014   | 1C05     |        |     |       | ldv  | \$r1,5         |      |      |
| 000026 | 0015   | 0001     |        |     |       | mcl  |                |      |      |
| 000027 | 0016   | 0C08     |        |     |       | dc   | x'0C08'        |      |      |
| 000028 | 0017   | 199A     |        |     |       | bnez   | \$r1,>aa       |      |      |
| 000029 |        |          |        |     | *     | \$USOUT  | !SLEW_2,=31,=1 |      |      |
| 000030 |        |          |        |     | *     | LDB  | \$B4,!SLEW_2   |      |      |
| 000031 | 0018   | CBC0     | 0031   |     |       | lab  | \$b4,slew_2    |      |      |
| 000032 | 001A   | E870     | 001F   |     |       | ldr  | \$r6,=31       |      |      |
| 000033 | 001C   | F870     | 0001   |     |       | ldr  | \$r7,=1        |      |      |
| 000034 | 001E   | 0001     |        |     |       | mcl  |                |      |      |
| 000035 | 001F   | 0801     |        |     |       | dc   | x'0801'        |      |      |
| 000036 |        |          |        |     | *     |  |                |      |      |
| 000037 |        | 0020     |        |     | nest  | equ  | \$             |      |      |
| 000038 | 0020   | CBC0     | 003F   |     |       | lab  | \$b4,buff_3    |      |      |
| 000039 | 0022   | E870     | 4011   |     |       | ldr  | \$r6,=Z'4011'  |      |      |
| 000040 | 0024   | 1C05     |        |     |       | ldv  | \$r1,5         |      |      |
| 000041 | 0025   | 0001     |        |     |       | mcl  |                |      |      |
| 000042 | 0026   | 0C08     |        |     |       | dc   | x'0C08'        |      |      |
| 000043 |        |          |        |     | *     |  |                |      |      |
| 000044 | 0027   | 198A     |        |     |       | bnez   | \$r1,>aa       |      |      |
| 000045 |        |          |        |     | *     |  |                |      |      |

Figure 8-3. Bound Unit TSTNOW

| TITLE  | TSTNOW    | 08/13/82 | 1334.4 | edt  | Fri    | HRS ASSEMBLER 9.00 | -LAF | PAGE | 0002                 |
|--------|-----------|----------|--------|------|--------|--------------------|------|------|----------------------|
| 000046 |           |          |        |      | *      | SUSOUT             |      |      | !SLEW 3,=20,=1       |
| 000047 |           |          |        |      | *      | LDB                |      |      | \$b4,!SLEW 3         |
| 000048 | 0028      | CBCD     | 0036   |      |        | lab                |      |      | \$b4,slew-3          |
| 000049 | 002A      | E870     | 0014   |      |        | ldr                |      |      | \$r6,=20             |
| 000050 | 002C      | F870     | 0001   |      |        | ldr                |      |      | \$r7,=1              |
| 000051 | 002E      | 0001     |        |      |        | mcl                |      |      |                      |
| 000052 | 002F      | 0801     |        |      |        | dc                 |      |      | x'0801'              |
| 000053 | 0030      | 1903     |        |      |        | bez                |      |      | \$r1,>bb             |
| 000054 |           |          |        |      | *      |                    |      |      |                      |
| 000055 | 0031      | A851     |        |      | aa     | ldr                |      |      | \$r2,=\$r1           |
| 000056 | 0032      | 0F82     |        |      |        | b                  |      |      | >cc                  |
| 000057 | 0033      | 2C00     |        |      | bb     | ldv                |      |      | \$R2,0               |
| 000058 |           |          |        |      | *      |                    |      |      |                      |
| 000059 |           |          |        |      | *Sc    | STRMRQ             |      |      |                      |
| 000060 |           | 0034     |        |      | cc     | equ                |      |      | \$                   |
| 000061 | 0034      | 0001     |        |      |        | mcl                |      |      |                      |
| 000062 | 0035      | 0103     |        |      |        | dc                 |      |      | x'0103'              |
| 000063 |           |          |        |      | *      |                    |      |      |                      |
| 000064 | 0036      | 2041     |        |      | slew   | dc                 |      |      | z'2041'              |
| 000065 | 0037      | 5B4E     | 4F57   | 5020 | buffer | text               |      |      | a'[NOW]'             |
| 000066 | 003A      | 2020     |        |      |        | resv               |      |      | 11,z'2020'           |
| 000067 | 0045      | 2020     |        |      | extra  | resv               |      |      | 5,z'2020'            |
| 000068 |           |          |        |      | *      |                    |      |      |                      |
| 000069 | 004A      | 2041     |        |      | slew_2 | dc                 |      |      | z'2041'              |
| 000070 | 004B      | 5B55     | 5345   | 5250 | buff_2 | text               |      |      | a'[USER]'            |
| 000071 | 004E      | 2020     |        |      |        | resv               |      |      | 12,z'2020'           |
| 000072 | 005A      | 2020     |        |      | extr_2 | resv               |      |      | 5,z'2020'            |
| 000073 | 005F      | 2041     |        |      | slew_3 | dc                 |      |      | z'2041'              |
| 000074 | 0060      | 5B6E     | 6F74   | 205B | buff_3 | text               |      |      | a'[not [equal 2 2]]' |
|        | 0063      | 6571     | 7561   | 6C20 |        |                    |      |      |                      |
|        |           | 3220     | 3250   | 5020 |        |                    |      |      |                      |
| 000075 | 0069      | 2020     |        |      |        | resv               |      |      | 12,z'2020'           |
| 000076 |           |          |        |      | *      |                    |      |      |                      |
| 000077 | 0075      | 0000     |        |      |        | end                |      |      | tstnow,begin         |
| 0000   | ERR COUNT |          |        |      |        |                    |      |      |                      |

Figure 8-3 (cont). Bound Unit TSTNOW

```

($S)GC086 MOD400-L3.0-07/02/0741
($S) "CLMIN...OR *"
($S) C?
GUIT
($S)                ** M4/3.0   V10.1 **
($S)THIS SYSTEM IS BUILT ON THE NEW DIRECTORY STRUCTURE
($S)DATE TIME (E.G.; 81 JAN 10 1405): 82 AUG 13 1538
($S)FRI AUG 13, 1982   15:38:08
($P)DAEMON GROUP READY!
($H)GROUP READY!
($H)$H
1 EC :CONSOLE
2 RDN
($S)RDY:
3 C :$H:
4 CWD :TERRY>LAF
($H)RDY:
5 CR TSTNOW.OK -REL -CISZ 4096 -LRSZ 512 -SZ 4
($H)RDY:
6 C :$S:
7 ATERRY>LAF>DEBUG
($S)DEBUG-R300-08/13/1514
8 NG
9 PG
($S) PG= 015003
10 SF ATERRY>LAF>TSTNOW.OK
11 SB TSTNOW
12 LB*
($S) SB TSTNOW
13 SP
($S)RDY:

14 ATERRY>LAF>TSTNOW
($S) *BU 1 $SL=001C $E=022263 + 0000 DATA=000000
15 AS X $E
16 DP X
($S)
($S) 022263/ CBCD 0036 E870 401C 1C05 0001 0C08 19AA ...6.Pa.....
17 SQ1 X+7 (AR)
18 SQ2 X+17 (AR;DH X+36/20;G0)
19 SQ3 X+27 (VH $R1;DH X+4A/20)

```

Figure 8-4. Debugging Session of TSTNOW

```

20 LG*
   ($S) QUICK BREAKPOINTS
   ($S) 1 LOC=02226A INST=19AA (AR;GO)
   ($S) 2 LOC=02227A INST=199A (AR;DH X+36/20;GO;GO)
   ($S) 3 LOC=02228A INST=198A (VH $R1;DH X+4A/20;GO)
21 GO
   ($S) FRI AUG 13, 1982 15:44:15
   ($S) OPERATOR.SYSTEM.OPR
   ($S) EC: 17 0805 (10)
   ($S) UNBALANCED QUOTATION MARKS, BRACKETS, OR PARENTHESES EXIST. THE NEXT FUNCTION
   ($S) IS PERFORMED. CORRECT THE DELIMITER AND RETRY.
   ($S) RDY:
22 DEBUG
   ($S) DEBUG - RDY
23 LG*
   ($S) QUICK BREAKPOINTS
   ($S) 1 LOC=02226A INST=19AA (AR;GO)
   ($S) 2 LOC=02227A INST=199A (AR;DH X+36/20;GO;GO)
   ($S) 3 LOC=02228A INST=198A (VH $R1;DH X+4A/20;GO)
24 LG*
25 LG*
   ($S) QUICK BREAKPOINTS
   ($S) 1 (AR;GO)
   ($S) 2 (AR;DH X+36/20;GO;GO)
   ($S) 3 (VH $R1;DH X+4A/20;GO)
26 RF GK
27 $H FO !LPT00
   ($S) RDY:
28 $H PR_LQK TSTNOW.GK
   ($S) RDY:
29 FG $H FO
   ($S) RDY:
30 FO !LPT00
31 DP 15003/500
32 FO
33 FG
   ($S) PG= 015003
34 RG
35 PG
   ($S) NO QUICK MEMORY EXISTS "PG"
36 GT
   ($S) RDY:

```

Figure 8-4 (cont). Debugging Session of TSTNOW

15. Assign the value of \$E to the temporary symbol X. Since the value of \$E is the base location of the bound unit, all subsequent references to a location in the bound unit can take the form: X + offset.
16. Dump one line of memory, starting at the location associated with the temporary symbol X.
17. Set quick breakpoint 1 and its associated directive line at offset 7 in the bound unit.
18. Set quick breakpoint 2 and its associated directive line at offset 17 in the bound unit.
19. Set quick breakpoint 3 and its associated directive line at offset 27 in the bound unit.
20. List all currently active quick breakpoints and their associated directive lines. Note that the directive line for quick breakpoint 2 ends with two GOs. The Debugger appends GO to the end of a quick breakpoint directive line, whether or not the user has already done so. The repetition of GO causes no problems. Once a GO is encountered in a directive line, the rest of the line (whatever it may be) is ignored.
21. Go from the bound unit breakpoint. The bound unit completes execution without any visible evidence that the quick breakpoints were encountered.
22. Reinvoke the Debugger.
23. List all quick breakpoints currently set and their associated directive lines.
24. Clear the quick breakpoints just listed.
25. List all quick breakpoints currently set and their associated directive lines. Note that although quick breakpoints 1, 2, and 3 are no longer set, their directive lines remain for future use. The clear breakpoint directive does not clear directive lines.
26. Close the quick disk file currently in use.
27. From the \$H group, change user-out to the line printer.
28. From the \$H group, invoke the Debugger utility PR\_QK to print the information written to the quick disk file TSTNOW.QK. The printout of this information is shown in Figure 8-5.
29. From the \$H group, change user-out back to its original device.

^TERRY>LAF>TSTNOW.QK  
\*\*\*DUMP OF DATA GENERATED BY MOD 400 MULTI-USER DEBUG QUICK BREAKPOINTS\*\*\*

DATA FROM MEMORY BUFFER 1  
QB1 ID= \$S TCB= 021E98 LEV= 1C \$R1=0000 \$R2=001C \$R3=0000 \$R4=0000 \$R5=0000 \$R6=001C \$R7=0000 \$B1=000000 \$B2=022704 \$B3=000  
00 \$B4=02229A \$B5=002F7C \$B6=022263 \$B7=023169 \$P=022268 \$I=3E00 \$S=401C  
QB2 ID= \$S TCB= 021E98 LEV= 1C \$R1=0000 \$R2=0013 \$R3=0000 \$R4=0000 \$R5=0000 \$R6=0013 \$R7=0000 \$B1=000000 \$B2=0226C4 \$B3=000  
00 \$B4=0222AE \$B5=002F7C \$B6=022263 \$B7=023169 \$P=022278 \$I=3E00 \$S=401C 022299/ 2041 4672 6920 4175 6720 3133 2C20 3139 3832  
2020 2020 3135 3A34 343A 3135 2020 2020 2020 2020 2041 4F50 4552 4154 4F52 2E53 5953 5445 402E 4F50 5220 2020  
QB3 ID= \$S TCB= 021E98 LEV= 1C \$R1=0805 0222AD/ 2041 4F50 4552 4154 4F52 2E53 5953 5445 402E 4F50 5220 2020 2020 2020 2020 2  
20 2020 2020 2020 2020 2041 5B6E 6F74 205B 6571 7561 6C20 3220 325D 5D20 2020  
EUF

8-85

CZ15-02

Figure 8-5. Contents of Quick Disk File TSTNOW.QK

30. Change the user-out of the (default) \$\$ group to the line printer. In this case, FO !LPT00 is a Debugger directive.
31. Dump 500 words of memory, starting at the location displayed earlier by the PQ directive as the start of quick memory (see step 9). A printout of this dump is shown in Figure 8-6.
32. Change user-out of the \$\$ group back to its original device.
33. Print the start of quick memory.
34. Return the quick memory block.
35. Print the start of quick memory. A message is returned verifying that the quick memory has been returned by the RQ directive.
36. Abort the Debugger from the \$\$ group.

### Sample Session 3

The code being debugged is in lower (system) memory, and deals with a semaphore related to all console I/O. The true breakpoint set in the following debugging session, shown in Figure 8-7, is subsequently encountered at each I/O request made to the console. Each numbered directive in Figure 8-7 is explained below by a correspondingly-numbered comment.

1. Change default group id to \$H.
2. Change working directory of \$H.
3. List all quick disk files in the directory.
4. Change default group id to \$\$\$. To use quick breakpoints, you must invoke the Debugger from the \$\$ group.
5. Change working directory of \$\$.
6. Establish standard I/O files for the \$\$ group.
7. Turn on the ready prompt. (Use of the ready prompt is optional. In this example, RDY helps to distinguish user input from system response.)
8. Invoke the Debugger.
9. Request quick memory, using the default values.
10. Print the start of quick memory.



```

015C03/ 0000 0800 0000 0100 5148 0001 6403 0001 6003 0100 0004 0004 0007 0001 0002 0007 .....QK..D...d.....
015C13/*0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 .....
015D03/ 0000 006E 5142 3120 4944 3020 2453 2054 4342 3020 3032 3145 3938 204C 4556 3020 ...MQB1 ID= $$ TCH= 021E98 LEV=
015D13/ 3143 2020 2020 2452 313D 3030 3030 2020 2452 323D 3030 3143 2020 2452 333D 3030 1C $R1=0000 $R2=001C $R3=00
015D23/ 3030 2020 2452 343D 3030 3030 2020 2452 353D 3030 3030 2020 2452 363D 3030 3143 00 $R4=0000 $R5=0000 $R6=001C
015D33/ 2020 2452 373D 3030 3030 2020 2442 313D 3030 3030 3030 2020 2442 323D 3032 3237 $R7=0000 $B1=000000 $B2=0227
015D43/ 3034 2020 2442 333D 3030 3030 3030 2020 2442 343D 3032 3232 3941 2020 2442 353D 04 $B3=000000 $B4=02229A $B5=
015D53/ 3030 3246 3743 2020 2442 363D 3032 3232 3633 2020 2442 373D 3032 3331 3639 2020 002F7C $B6=022263 $B7=023169
015D63/ 2450 3030 3232 3236 4220 2024 493D 3345 3030 2020 2453 3034 3031 4320 2020 2020 $P=02226B $I=3E00 $S=401C
015D73/*0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 .....
015E03/ 0000 00C4 5142 3220 4944 3020 2453 2054 4342 3020 3032 3145 3938 204C 4556 3020 ....QB2 ID= $$ TCH= 021E98 LEV=
015E13/ 3143 2020 2020 2452 313D 3030 3030 2020 2452 323D 3030 3133 2020 2452 333D 3030 1C $R1=0000 $R2=0013 $R3=00
015E23/ 3030 2020 2452 343D 3030 3030 2020 2452 353D 3030 3030 2020 2452 363D 3030 3133 00 $R4=0000 $R5=0000 $R6=0013
015E33/ 2020 2452 373D 3030 3030 2020 2442 313D 3030 3030 3030 2020 2442 323D 3032 3236 $R7=0000 $B1=000000 $B2=0226
015E43/ 4334 2020 2442 333D 3030 3030 3030 2020 2442 343D 3032 3232 4145 2020 2442 353D C4 $B3=000000 $B4=0222AE $B5=
015E53/ 3030 3246 3743 2020 2442 363D 3032 3232 3633 2020 2442 373D 3032 3331 3639 2020 002F7C $B6=022263 $B7=023169
015E63/ 2450 3030 3232 3237 4220 2024 493D 3345 3030 2020 2453 3034 3031 4320 2020 2020 $P=02227B $I=3E00 $S=401C
015E73/ 3032 3232 3939 2F20 3230 3431 2034 3637 3220 3639 3230 2034 3137 3520 3637 3230 022299/ 2041 4672 6920 4175 6720
015E83/ 2033 3133 3320 3243 3230 2033 3133 3920 3338 3332 2032 3032 3020 3230 3230 2033 3133 2C20 3139 3832 2020 2020 3
015E93/ 3133 3520 3341 3334 2033 3433 4120 3331 3335 2032 3032 3020 3230 3230 2032 3032 135 3A34 343A 3135 2020 2020 202
015EA3/ 3020 3230 3230 2032 3032 3020 3230 3431 2034 4635 3020 3435 3532 2034 3135 3420 0 2020 2020 2041 4F50 4552 4154
015EB3/ 3446 3532 2032 4535 3320 3539 3533 2035 3434 3520 3444 3245 2034 4635 3020 3532 4F52 2E53 5953 5445 4D2E 4F50 52
015EC3/ 3230 2032 3032 3020 2020 2020 0000 0000 0000 0000 0000 0000 0000 0000 0000 .....
015ED3/*0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 .....
015F03/ 0000 006D 5142 3320 4944 3020 2453 2054 4342 3020 3032 3145 3938 204C 4556 3020 ...MQB3 ID= $$ TCH= 021E98 LEV=
015F13/ 3143 2020 2020 2452 313D 3038 3035 2020 2020 3032 3232 4144 2F20 3230 3431 2034 1C $R1=0805 0222AD/ 2041 4
015F23/ 4635 3020 3435 3532 2034 3135 3420 3446 3532 2032 4535 3320 3539 3533 2035 3434 F50 4552 4154 4F52 2E53 5453 544
015F33/ 3520 3444 3245 2034 4635 3020 3532 3230 2032 3032 3020 3230 3230 2032 3032 3020 5 4D2E 4F50 5220 2020 2020 2020
015F43/ 3230 3230 2032 3032 3020 3230 3230 2032 3032 3020 3230 2032 3032 3020 3230 2020 2020 2020 2020 2020 2020 2020
015F53/ 3230 2032 3034 3120 3542 3645 2036 4637 3420 3230 3542 2036 3537 3120 3735 3631 20 2041 5B6E 6F74 205B 6571 7561
015F63/ 2036 4332 3020 3332 3230 2033 3235 4420 3544 3230 2032 3032 3020 2020 2020 0000 6C20 3220 325D 5D20 2020 ..
015F73/*0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 .....
0160F3/ 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 .....

```

Figure 8-6. Dump of Quick Memory

```

($S)GC096 MOD400-L3.0-07/02/0741
($S) "CLMIN...OR *"
($S) C?
QUIT
($S)          ** M4/3.0   V10.1 **
($S)THIS SYSTEM IS BUILT ON THE NEW DIRECTORY STRUCTURE
($S)DATE TIME (E.G., 81 JAN 10 1405):  82 AUG 13 1557
($S)FRI AUG 13, 1982    15:57:08
($P)DAEMON GROUP READY!
($H)GROUP READY!
($H)$H
1  C :$H:
2  CWD ^TERRY>LAF
   ($H)RDY:
3  LS *.GK -BF
   ($H)

   DIRECTORY: ^TERRY>LAF
   ($H)
   ($H)  TSTNOW.GK      R      64
   ($H)  SAMPLE.GK     R      48
   ($H)
   ($H) TOTAL SECTORS      112
   ($H)RDY:
4  C :$S:
5  CWD ^TERRY>LAF
6  EC !CONSOLE
7  RDN
   ($S)RDY:
8  DEBUG
   ($S)DEBUG-R300-08/13/1514
9  MG
10 PQ
   ($S) PQ= 015C03
11 SF SAMPLE.GK
12 DP 131D
   ($S)
   ($S) 00131D/ CBCD FF52 D3CD 1994 D3CD 20C8 D3CD 1C5E ...R..... ^
13 SG3 131D (AR)

```

8-88

CZ15-02

Figure 8-7. Debugging Session (Example 3)

```
14 LG*
   ($S) QUICK BREAKPOINTS
   ($S) 3 LOC=00131D INST=CBCD (AR;GO)
15 $H LSR
   ($H)^TERRY>LAF
   ($H)^KB>SYSLIB1
   ($H)^KB>SYSLIB2
   ($H)RDY:
16 $H LWD
   ($H)^TERRY>LAF
   ($H)RDY:
17 CG*
18 LG*
   ($S) QUICK BREAKPOINTS
   ($S) 3 (AR;GO)
19 RF GK
20 RG
21 FG
   ($S) NO QUICK MEMORY EXISTS "PG"
22 GT
   ($S)RDY:
23 $H FO !LPT00
   ($H)RDY:
24 $H PR_GK SAMPLE.GK
   ($H)RDY:
```

Figure 8-7 (cont). Debugging Session (Example 3)

11. Open the quick disk file SAMPLE.QK.
12. Dump a line of memory, starting at location 131D. This location is in system memory.
13. Set quick breakpoint 3 at location 131D, specifying a directive line. As already mentioned, breakpoint 3 will be encountered at each I/O request made to the console.
14. List all quick breakpoints currently set and their associated directive lines.
15. List search rules for the \$H group.
16. List the current working directory for the \$H group. Even though the Debugger is running in the \$S group, a task running in \$H (steps 15 and 16) can encounter a true breakpoint.
17. Clear all quick breakpoints. The Debugger has remained active in the \$S group while quick breakpoints were encountered by a task running in the \$H group (steps 15 and 16). Input to the \$S group is still being handled by the Debugger.
18. List all quick breakpoints currently set and their associated directive lines. Note that no quick breakpoints are currently set. There is, however, a directive line ready to be used for quick breakpoint 3. The CQ directive does not clear directive lines.
19. Close the currently active quick disk file.
20. Return the quick memory requested by the MQ directive (step 8).
21. Print the start of quick memory. The message shows that quick memory has been returned.
22. Abort the Debugger from the \$S group.
23. Change user-out for the \$H group to the line printer.
24. From the \$H group use the Debugger utility PR\_QK to print information stored in the quick disk file. A print-out of this information is shown in Figure 8-8.

8-91

^TERRY>LAF>SAMPLE.QK  
\*\*\*DUMP OF DATA GENERATED BY MOD 400 MULTI-USER DEBUG QUICK BREAKPOINTS\*\*\*

```
DATA FROM MEMORY BUFFER 1
QB3 ID= $$ TCB= 000777 LEV= 1B $R1=0000 $R2=009E $R3=0009 $R4=0002 $R5=0001 $R6=2453 $R7=0002 $B1=0081F6 $B2=008403 $B3=001
A5 $B4=0231D5 $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $$ TCB= 000777 LEV= 1B $R1=0000 $R2=00FE $R3=0002 $R4=002A $R5=0023 $R6=2453 $R7=008C $B1=0086F7 $B2=001237 $B3=000
77 $B4=0011A6 $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $$ TCB= 000777 LEV= 1B $R1=0000 $R2=00FE $R3=0003 $R4=0001 $R5=00F7 $R6=0014 $R7=4101 $B1=0083E6 $B2=001221 $B3=000
77 $B4=0011CB $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $$ TCB= 000777 LEV= 1B $R1=0000 $R2=0000 $R3=0002 $R4=0003 $R5=0023 $R6=0014 $R7=0000 $B1=0082F2 $B2=0011CB $B3=023
F5 $B4=0233F5 $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $$ TCB= 000777 LEV= 1B $R1=0000 $R2=00FE $R3=0003 $R4=0001 $R5=00F7 $R6=0022 $R7=4101 $B1=0082B3 $B2=001221 $B3=000
77 $B4=0011CB $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $$ TCB= 000777 LEV= 1B $R1=0000 $R2=0000 $R3=0002 $R4=0003 $R5=0023 $R6=0022 $R7=0000 $B1=008625 $B2=0011CB $B3=023
F5 $B4=0233F5 $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $$ TCB= 000777 LEV= 1B $R1=0000 $R2=009E $R3=0009 $R4=0002 $R5=0001 $R6=0022 $R7=0002 $B1=0086CD $B2=008403 $B3=001
A5 $B4=0231D5 $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B

DATA FROM MEMORY BUFFER 2
QB3 ID= $$ TCB= 000777 LEV= 1B $R1=0000 $R2=00FE $R3=0002 $R4=0052 $R5=0023 $R6=2448 $R7=008C $B1=00846C $B2=001237 $B3=000
77 $B4=0011A6 $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $$ TCB= 000777 LEV= 1B $R1=0000 $R2=0044 $R3=0009 $R4=0009 $R5=0001 $R6=2448 $R7=0002 $B1=00864F $B2=0086CD $B3=001
A5 $B4=040623 $B5=001313 $B6=0001A7 $B7=021A3C $P=00131E $I=3E04 $S=401B
QB3 ID= $$ TCB= 000777 LEV= 1B $R1=0000 $R2=00FE $R3=0003 $R4=0001 $R5=00F7 $R6=000B $R7=4102 $B1=00038D $B2=001221 $B3=000
77 $B4=0011CB $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $$ TCB= 000777 LEV= 1B $R1=0000 $R2=0000 $R3=0002 $R4=0003 $R5=0023 $R6=000B $R7=0000 $B1=0087DE $B2=0011CB $B3=023
15 $B4=023215 $B5=001313 $B6=0001A7 $B7=022A3C $P=00131E $I=3E04 $S=401B
QB3 ID= $$ TCB= 000777 LEV= 1B $R1=0000 $R2=00FE $R3=0003 $R4=0001 $R5=00F7 $R6=000C $R7=4102 $B1=00042D $B2=001221 $B3=000
77 $B4=0011CB $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $$ TCB= 000777 LEV= 1B $R1=0000 $R2=0000 $R3=0002 $R4=0003 $R5=0023 $R6=000C $R7=0000 $B1=00892E $B2=0011CB $B3=023
15 $B4=023215 $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $$ TCB= 000777 LEV= 1B $R1=0000 $R2=00FE $R3=0003 $R4=0001 $R5=00F7 $R6=000C $R7=4102 $B1=00868E $B2=001221 $B3=000
77 $B4=0011CB $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
```

Figure 8-8. Dump of Quick Disk File Sample .QK

CZ15-02

```

DATA FROM MEMORY BUFFER 3
QB3 ID= $S TCB= 000777 LEV= 1B $R1=0000 $R2=0000 $R3=0002 $R4=0003 $R5=0023 $R6=000C $R7=0000 $B1=008331 $B2=0011CB $B3=023
15 $B4=023215 $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $S TCB= 000777 LEV= 1B $R1=0000 $R2=0000 $R3=0008 $R4=0009 $R5=01A8 $R6=2448 $R7=0001 $B1=00044A $B2=021E83 $B3=001
A7 $B4=04062D $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $S TCB= 000777 LEV= 1B $R1=0000 $R2=00FE $R3=0003 $R4=0001 $R5=00F7 $R6=0006 $R7=4101 $B1=0084AB $B2=001221 $B3=000
77 $B4=0011CB $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $S TCB= 000777 LEV= 1B $R1=0000 $R2=0000 $R3=0002 $R4=0003 $R5=0023 $R6=0006 $R7=0000 $B1=00857D $B2=0011CB $B3=023
15 $B4=023215 $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $S TCB= 000777 LEV= 1M $R1=0000 $R2=009E $R3=0009 $R4=0002 $R5=0001 $R6=0006 $R7=0002 $B1=00874B $B2=0086CD $B3=001
A5 $B4=022015 $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $S TCB= 000777 LEV= 1B $R1=0000 $R2=00FE $R3=0002 $R4=0044 $R5=0023 $R6=2448 $R7=008C $B1=0081E1 $B2=001237 $B3=000
77 $B4=0011A6 $B5=001313 $B6=0001A7 $B7=021C3C $P=00131E $I=3E04 $S=401B
QB3 ID= $S TCB= 000777 LEV= 1B $R1=0000 $R2=0044 $R3=0009 $R4=0009 $R5=0001 $R6=2448 $R7=0002 $B1=008496 $B2=0086CD $B3=001
A5 $B4=040623 $B5=001313 $B6=0001A7 $B7=021A3C $P=00131E $I=3E04 $S=401B

```

```

DATA FROM MEMORY BUFFER 4
QB3 ID= $S TCB= 000777 LEV= 1B $R1=0000 $R2=00FE $R3=0003 $R4=0001 $R5=00F7 $R6=0008 $R7=4102 $B1=008904 $B2=001221 $B3=000
77 $B4=0011CB $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $S TCB= 000777 LEV= 1B $R1=0000 $R2=0000 $R3=0002 $R4=0003 $R5=0023 $R6=0008 $R7=0000 $B1=0088EF $B2=0011CB $B3=023
15 $B4=023215 $B5=001313 $B6=0001A7 $B7=022A3C $P=00131E $I=3E04 $S=401B
QB3 ID= $S TCB= 000777 LEV= 1B $R1=0000 $R2=0000 $R3=0008 $R4=0009 $R5=01A8 $R6=2448 $R7=0001 $B1=008331 $B2=021E83 $B3=001
A7 $B4=04062D $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $S TCB= 000777 LEV= 1B $R1=0000 $R2=00FE $R3=0003 $R4=0001 $R5=00F7 $R6=0006 $R7=4101 $B1=00864F $B2=001221 $B3=000
77 $B4=0011CB $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B

```

```

QB3 ID= $S TCB= 000777 LEV= 1B $R1=0000 $R2=0000 $R3=0002 $R4=0003 $R5=0023 $R6=0006 $R7=0000 $B1=008385 $B2=0011CB $B3=023
15 $B4=023215 $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $S TCB= 000777 LEV= 1B $R1=0000 $R2=009E $R3=0009 $R4=0002 $R5=0001 $R6=0006 $R7=0002 $B1=00810F $B2=0086CD $B3=001
A5 $B4=022015 $B5=001313 $B6=0001A7 $B7=0007F8 $P=00131E $I=3E04 $S=401B
QB3 ID= $S TCB= 000777 LEV= 1B $R1=0000 $R2=00FE $R3=0002 $R4=002A $R5=0023 $R6=2453 $R7=008C $B1=0083EE $B2=001237 $B3=000
77 $B4=0011A6 $B5=001313 $B6=0001A7 $B7=021C3C $P=00131E $I=3E04 $S=401B
EOF

```

Figure 8-8 (cont). Dump of Quick Disk File Sample .QK

## *Section 9*

# **REQUESTING AND USING MEMORY DUMPS**

This section provides procedures for displaying, printing, and analyzing the contents of main memory.

### OVERVIEW

The Create Volume (CV) command allows disks to be prepared to hold main memory and communications controller memory dumps.

There are two methods to process memory dumps, each geared to a specific need. DPEDIT is a Dump Edit utility that allows you to print a main memory dump. XRAY is an Interactive Memory Dump Editor utility that allows you to display all or a portion of main memory. (Contact your Honeywell support representative for information on displaying a communications controller memory dump.) This section tells how to:

- Prepare a volume to which dump utilities called DMPMEM write the contents of main memory and, optionally, communications controller memory
- Execute the dump utilities, writing the contents of main memory (and/or communications controller memory) to your dump volume
- Print and analyze a main memory image, using the DPEDIT utility

- Display and analyze a main memory image, using the XRAY utility.

The general procedure for taking memory dumps is to:

1. Prepare a disk dump volume, using the Create Volume command with the -MDUMP and/or -COMMFILE argument. Once prepared, the volume contains empty files whose simple pathname, assigned by the system, is DUMPFIL corresponding to the -MDUMP argument and COMMFILE corresponding to the -COMMFILE argument. The volume also contains a file named ZVDLDR in the root directory and a directory named ZVPROG DIR, which contains several other files. Collectively, these files comprise the dump utilities called DMPMEM. Execution of the dump utilities write an image of memory.

The dump utilities are loaded and executed independently of the Executive. They can therefore be used to obtain an image of memory after a system failure. You should, of course, prepare a dump volume while the system is operative, before it is needed.

2. Mount the dump volume; load and execute the dump utilities.
3. On a system running under the Executive, invoke either the XRAY or DPEDIT utility. DPEDIT is used to print DUMPFIL in edited or unedited format. (DPEDIT can also be used to print the contents of current main memory.) XRAY is used to display DUMPFIL in edited or unedited format on your terminal. (XRAY can also be used to display the contents of current main memory.)

### USING THE DUMP UTILITIES

You can dump the contents of main memory to the same volume from which you initialize the system, or you can dump memory to a different volume. Since the need to take a memory dump is infrequent, you will probably want to dump to a volume different from your system disk. This avoids reserving space on your system disk for the dump file(s). If your system has only one disk and it is a fixed disk, you can chose to place the dump files(s) on multiple diskettes or to share space on the system disk.

The following paragraphs first describe taking a dump with a volume other than the system volume; mutiple-diskette and shared volumes are discussed afterwards.



## Creating a Dump Volume

The volume which is to serve as a dump volume must have been previously formatted (via the Create Volume command with the -FT argument). It need not be empty, as long as any existing files on the volume allow space for DUMPFIL and, optionally, COMMFILE to be created in the volume root directory.

To create a dump volume, mount a formatted volume and enter the command:

```
CV pathname -MDUMP nnnn [-COMMFILE nn]
```

pathname

Designates the disk volume that you have mounted. This argument can take the form:

```
!device_name>volume_id
```

or

```
!device_name
```

Device name is the symbolic device name assigned to the drive on which the volume is mounted. If you use the first format, specifying both device name and volume identifier, the system checks the label on the mounted volume, thus ensuring that Create Volume runs against the correct volume.

If the drives of your system are not labelled with their device names, a simple way of determining a device name is by using the DRIVE active function. Suppose, for example, that you have mounted a volume DMPVOL on an unlabelled drive. Using the DRIVE active function, enter:

```
CV [DRIVE ^DMPVOL] -MDUMP nnnn
```

This command line resolves to:

```
CV !RCD00 -MDUMP nnnn
```

assuming that the device name of the unlabelled drive is RCD00. For further information on this active function, see the Commands manual.

-MDUMP nnnn

This argument writes the dump utilities to the specified volume and creates a file in the volume root directory, named DUMPFIL, of the size nnnn, where nnnn is a decimal number of 4096-word units. (The argument -MDUMP 20, for example, specifies a file size of 20 x 4096 or 81,920 words.)

There is no default value for nnnn; you must supply one. The minimum value for nnnn is 16 (65,536 words). Entering a value less than 16 does not cause an error; instead, Create Volume creates a dump file of 65,536 words as if you had entered 16. The maximum allowable value for nnnn depends upon the disk medium that you select (e.g., mass storage module, removable cartridge) for a dump volume. The maximum values for each medium are given later in this section under "Maximum Dumpfile Size."

#### -COMMFILE nn

This argument writes the dump utilities to the specified volume and creates a file in the volume root directory, named COMMFILE, of the size nn, where nn is a decimal number of 4096-word units. (The argument -COMMFILE 8, for example, specifies a file size of 8 x 4096 or 32,768 words.)

#### SETTING DUMPFILe SIZE

Your dumpfile should be at least equal in size to the main memory installed in your system; it may be greater in size as well.

The size of installed memory can be determined by entering the command VIDEO. The first item of information displayed by this command is the amount of memory occupied by the Executive and the total amount of installed memory. This information is displayed in the following format:

237K/768K

In this example, the Executive occupies 237K words of memory; the size of installed memory is 768K words. Dividing 768 by 4 gives you the smallest value you can enter for nnnn (192) when preparing to dump 768K words of memory.

Most current communications controllers contain 64K bytes (32K words) of memory.

#### DUMPFILe FORMAT

Both DUMPFILe and COMMFILE are a fixed-relative files, immediately subordinate to the volume root directory. That is, there pathname is ^vol\_id>DUMPFILe or ^vol\_id>COMMFILE.

## DETERMINING AVAILABLE DISK SPACE

It is possible that a used volume cannot accommodate a dump file large enough for your purposes. You can determine the availability of space on a volume by means of the Get Quota (GQ) command, specifying the pathname of the volume as a device name. For example, if your ^DMPVOL is mounted on the drive called RCD00, enter:

```
GQ !RCD00
```

This utility returns the following information relevant to creating a dump file:

- Total capacity of the volume, in sectors
- Number of sectors used.

By subtracting the second item from the first, you arrive at the amount of disk space available.

On a disk, a sector holds 256 bytes. The amount of memory (in words) that the disk will hold is the number of words per sector times the number of sectors (e.g., 128 words x number of sectors available).

## MAXIMUM DUMPFILe SIZE

The largest dumpfiles you can create on various disk media are shown below. After each maximum size (given in words) appears the corresponding value for the -MDUMP nnnn argument.

| <u>Medium</u>                | <u>Maximum<br/>Size</u> | <u>-MDUMP<br/>nnnn</u> | <u>Percent<br/>of Disk</u> |
|------------------------------|-------------------------|------------------------|----------------------------|
| diskette, single volume      | 192,512                 | 47                     | 100                        |
| mass storage module (67 MB)  | 33,554,432              | 8192                   | 96                         |
| mass storage module (256 MB) | 33,554,432              | 8192                   | 24                         |
| removable cartridge module   | 6,737,920               | 1645                   | 100                        |
| removable Lark cartridge     | 10,436,608              | 2548                   | 100                        |
| fixed disk (132 MB)          | 33,554,432              | 8192                   | 56                         |
| fixed disk (142 MB)          | 33,554,432              | 8192                   | 48                         |
| fixed disk (295 MB)          | 33,554,432              | 8192                   | 20                         |
| fixed disk (413 MB)          | 33,554,432              | 8192                   | 12                         |

## MULTIPLE-VOLUME DISKETTE

To create a multiple-volume dumpfile on diskette that is suitable for dumping memory, it is necessary to format each diskette as a member of a multiple-volume set. To create a multiple-volume diskette set, use the CV command as follows:

```
CV !DSKnn -FT name  
CV !DSKnn -SMS setname -MBR nn
```

where setname and member number are as described under the CV command description in the Commands manual. These commands should be used for each diskette in the set. Each diskette will hold 192K words of memory, so the total number of diskettes required is the main memory size (plus any optional communications controller memory) divided by 192K words. The first diskette in the set should be member number 1, the second should be 2, and so on.

The number of diskettes required to hold representative memory sizes are shown below. After each memory size (given in words) appears the number of required diskettes.

| <u>Memory Size</u> | <u>Required Diskettes</u> |
|--------------------|---------------------------|
| 2M                 | 11                        |
| 4M                 | 22                        |
| 8M                 | 44                        |

When all the diskettes have been formatted, the DUMPFILe should be put on each diskette. To put this on each diskette, use the CV command as follows:

```
CV !DSKnn -MDUMP nnnn -MBR nn
```

The -MBR argument of the CV command will prevent the unnecessary copying of dump utility programs and files to all but the first diskette. If communications controllers are also being dumped, the -COMMFILe argument of the CV command should be used with the first diskette. If a large COMMFILe is to be created, it may be necessary to use a smaller DUMPFILe (as specified by the -MDUMP argument) on the first diskette. Note that the DUMPFILe on the first diskette must be at least 64K words (-MDUMP 16).

When taking a memory dump with a multiple-volume diskette set, the first diskette should be placed in the drive and booted. After the first diskette's DUMPFILe is filled, the dump utility will issue a prompt to mount the next diskette. This procedure will continue until the end of memory is reached, at which point the dump utility will issue a prompt to remount the first diskette. This signals the end of the dump and the first diskette has to be remounted.

Before the dump can be analyzed, the individual dumpfiles must be combined into one dumpfile. This is done by using the Copy command to copy each of the diskettes onto a higher capacity disk device. To copy the diskettes, use the CP command as follows:

```
CP ^name>DUMPFILe DUMPFILe -APPEND
```

When all of the diskettes have been combined in this fashion, the resulting DUMPFILe can be analyzed.

## Shared Dump and System Volumes

As mentioned at the outset of this section, it is possible to create a disk dump volume that serves also as a system volume used to initialize the system. To create such a "shared" volume, enter the command line:

```
CV pathname -BOOT [X'cccn'] -MDUMP nn  
-BOOT [X'cccn']
```

Entering the -BOOT argument writes a bootstrap and intermediate loader records to sectors 0 through 6 of the volume identified by the pathname argument. The optional value X'cccn' allows you to place the bootstrap and intermediate loader records on a different volume from the Executive.

When using this directive with cartridge disks you should be aware that the bootstrap and intermediate loader records must reside on a removable disk.

The hexadecimal digits ccc specify the channel number associated with the volume that holds the Executive (e.g., if the Executive disk is on channel 1400, ccc = 140).

The value of n can be zero or one. Zero indicates that the volume holding the Executive is removable disk or a Winchester technology fixed disk; the value one indicates a fixed cartridge or cartridge module disk. If the removable cartridge or cartridge module disk (specified by pathname) and the fixed cartridge or cartridge module disk (signified by X'cccl') reside on the same device and so share the same channel number, the value of ccc can be zero. When initializing the system, you enter the shared channel number.

-MDUMP nnnn

For a discussion of this argument, see the preceding subsection "Creating a Dump Volume".

You may create the memory dump files (DUMPPFILE and COMMPFILE) on the present boot device of a single disk system by using the following CV command:

```
CV ^volname -MDUMP nnnn -COMMPFILE nn
```

This command will write the bootstrap records and assume that the necessary memory dump file and directory (ZVDLDR and ZVPROG\_DIR) exist in the volume's root directory.

## Taking a Dump Using a Control Panel

The following procedure applies to systems that support a control panel. The dump volume used can be one prepared with or without the Create Volume -BOOT argument. In either case, it must be one prepared with the Create Volume -MDUMP argument or -COMMFIL argument, or both.

The memory dump will be more useful if the contents of the registers are recorded before taking the dump. Once you perform the memory dump, keep the register information with the dump.

To write an image of main memory (and/or communications controller memory) to a dump file, first mount the volume containing a dumpfile and the dump utilities. Then perform the following steps at the system's control panel:

1. Press STOP and CLEAR.
2. Set P register to 004. This is done by:
  - a. Pressing SELECT and entering the register code E0 from the control panel key pad.
  - b. Pressing CHANGE and entering the value 004.

This step instructs the central processor not to clear memory and to start execution at the address that you enter in register B1 (step 4).

3. Enter in Register D1 (R1) the channel number of the device on which you have mounted the dump or dump/system volume. This is done by:
  - a. Pressing SELECT and entering the register code D1.
  - b. Pressing CHANGE and entering the channel number. If the volume is shared (i.e., was created with the -BOOT and -MDUMP arguments), change the last digit of the channel number to the value 8. That is, set R1 to the value X'ccc8', where 'ccc' represents the first three digits of the channel number. The value 8 causes the BTDUMP routine, rather than the system bootstrap record, to be loaded into memory.
4. Enter in register B1 the starting memory address (hexadecimal) into which the MDUMP routine is to be loaded. This address should be at least 2000 (hexadecimal) to ensure that hardware-dedicated locations in memory (e.g., trap save area pointers, interrupt vectors) are not overlaid. The contents of B1 are set by:
  - a. Pressing SELECT and entering B1 in the Location register.

- b. Pressing CHANGE and entering a hexadecimal address not less than 2000.
5. Press LOAD, then EXECUTE. During successful execution of the memory dump, the contents of B1 increment until reaching the last address of installed memory. Then B1 is cleared to zeros.

#### Taking a Dump Using the System Control Facility (SCF)

The following procedure applies to taking dumps on a system using the System Control Facility (SCF). This facility allows the operator's terminal to be used as a control panel. The legends STEP, SELECT, CHANGE, LOAD, and EXECUTE, referred to in the procedure, appear on an overlay that fits over the function keys of your terminal. In case your terminal is not equipped with this keyboard overlay, each reference is followed parenthetically by a function key number (e.g., STEP (F6)). In the absence of a keyboard overlay showing STEP, press the function key labeled F6.

The dump volume used can be one prepared with or without the Create Volume -BOOT argument. In either case, it must be one prepared with the Create Volume -MDUMP argument or -COMMFIL argument, or both.

The memory dump will be more useful if the contents of the registers are recorded before taking the dump. Once you perform the memory dump, keep the register information with the dump.

To write an image of main memory (and/or communications controller memory) to a dump file, first mount the volume containing a dumpfile and the dump utilities. Then perform the following steps at the operator's terminal.

1. Ensure that the terminal is in panel mode. The current mode is the last item of information displayed on the 25th line of the monitor screen. To put the terminal in panel mode, type:  
  
ESC ESC # C/R ENA PAN
2. Press STEP (F6) and CLEAR.
3. Set P register to 004. This is done by:
  - a. Pressing SELECT (F2) and typing the register code E0.
  - b. Pressing CHANGE (F3) and typing the value 004.

This step instructs the central processor not to clear memory and to start execution at the address you enter in register B1 (step 5).

4. Enter in Register D1 (R1) the channel number of the device on which you have mounted the dump or dump/system volume. This is done by:
  - a. Pressing SELECT (F2) and typing the register code D1.
  - b. Pressing CHANGE (F3) and typing the channel number. If the volume is shared (i.e., was created with the -BOOT and -MDUMP arguments), change the last digit of the channel number to the value 8. That is, type the value X'ccc8', where 'ccc' represents the first three digits of the channel number. The value 8 causes the BTDUMP routine, rather than the system bootstrap record, to be loaded into memory.
5. Enter in register B1 the starting memory address (hexadecimal) into which the MDUMP routine is to be loaded. This address should be at least 2000 (hexadecimal) to ensure that hardware-dedicated locations in memory (e.g, trap save area pointers, interrupt vectors) are not overlaid. The contents of B1 are set by:
  - a. Pressing SELECT (F2) and typing B1.
  - b. Pressing CHANGE (F3) and typing a hexadecimal address not less than 2000.
6. Press LOAD (F1), then EXECUTE (F12). During successful execution of the memory dump, the contents of B1 increment until reaching the last address of installed memory. Then B1 is cleared to zeros.

#### USING DPEDIT

The DPEDIT utility can be used to print the contents of a memory dump. Printed dumps produced by the DPEDIT utility are written to the user-out file, which must be capable of receiving a 132-character line. There are two sources of dumps:

- Files created by the previous execution of the BTDUMP utility. (All or selected portions of the file can be dumped.)
- Main memory. (A dump of main memory allows you to determine the configuration under which DPEDIT is executing.)

Dumps produced by DPEDIT may be logical (edited format) dumps or physical (memory image format) dumps. Control arguments in the DPEDIT command (described later in this section) allow you to request either a logical or physical dump. If these control arguments are omitted, execution of DPEDIT produces a full logical dump followed by a full physical dump.



Logical and physical dumps are printed in both hexadecimal and ASCII notation. Duplicate lines, if any, are suppressed. Suppressed lines are described under "Line Format."

### Page Header

The page heading contains the following information:

- The origin of the dump (main memory or a dump file)
- The date and time of the edit
- The version of DPEDIT used
- The version of the system DPEDIT is executing on
- The pool and group currently being dumped (for a logical dump)
- The page number.

### Line Format

The format of a basic line for both logical and physical dumps is as follows:

| <u>Columns</u> | <u>Content</u>  |
|----------------|---|
| 1-6            | Six hexadecimal digits designating the starting physical (real) address of the line of dump information. The hexadecimal digit in print position 6 is always 0. This forces the dump line to agree with the template printed at the heading of each page. |
| 7-8            | Slash (/) followed by a space.  |
| 9-14           | Six hexadecimal digits designating the starting virtual address of the line of dump information.  |
| 15-17          | Slash (/) followed by two spaces.   |
| 18-98          | Sixteen consecutive words. Each word is represented by four hexadecimal digits and is followed by a space.  |
| 99             | Space   |
| 100-131        | ASCII representation of the previous group of 16 consecutive words. A byte that is non-printing is designated by a period (.).  |

Duplicate lines within the dumps are shown as:

|     |        |
|-----|--------|
| 1-8 | Spaces |
|-----|--------|

Columns

Content

9-95 \* \* \* \* \* (indicates one or more duplicate lines)  
96-131 Spaces.

Physical Dump

In a physical dump, the leftmost six columns of data designate real memory addresses. When the Memory Management Unit (MMU) is in use, there may be ranges of invalid virtual addresses in a physical dump from main memory. When an invalid virtual address is encountered, a message interrupts the listing of memory locations, specifying the invalid virtual address and the physical address for which no valid virtual address exists.

The virtual address is displayed whenever possible. If it does not appear, it means that the virtual and physical addresses are the same (in low memory), or that DPEDIT could not discover the virtual address corresponding to a given physical address. The listing of real locations resumes when the valid virtual address is known. The numerical sequence of real memory addresses, before and after the message, is unbroken.

A physical dump from an external dump file does not display invalid virtual address messages, and the left column of addresses is an uninterrupted continuum of physical addresses.

Logical Dump

By means of DPEDIT control arguments, the user can select the task groups about which a logical dump supplies information. File system information can also be selected.

The main addresses in a logical dump are virtual addresses (columns 9-14). The leftmost six columns of data are physical addresses, and are displayed whenever they differ from the virtual addresses. This applies to dumps of disk files as well as to dumps of main memory. For disk files, Dump Edit calculates the virtual address in the same way as the Memory Management Unit would under the same conditions.

The arrangement of information in a logical dump is described in the following paragraphs.

SYSTEM SUMMARY

The information contained in a logical dump includes:

- Location and contents of hardware-dedicated main storage
- System time of dump

- Time of system boot
- Time of power-fail restart (if it occurred)
- Hardware configuration
- Location and contents of System Control Block (SCB)
  - Model number of central processor
  - Presence (or absence) of the Commercial Instruction Processor, the Scientific Instruction Processor, and the Memory Management Unit
  - Value of the real-time clock scan cycle
  - Presence (or absence) of an operator's terminal
  - High address of physical memory.
- Software configuration
  - Name and version of operating system
  - Presence (or absence) of the system message library
  - Size of trap save area (TSA)
  - Size of interrupt save area (ISA)
  - Number of indirect request blocks (IRBs) in IRB pool
  - Presence (or absence) of the batch task group.
- Memory pool data
  - Pool identification
  - Starting address of pool
  - End address of pool
  - Total size of pool
  - Physical start address
  - Total available space
  - Maximum contiguous available space
  - Number of available fragments (pieces) of pool space
  - Number of users
  - Table of attributes for each pool.
- Additional pool information
  - Memory pool descriptor
  - Bit map (unless it is a queue-managed pool)
  - Segment descriptors.
- System symbol table

The names and values of all symbols that have an entry in the system symbol table are displayed. Symbols are grouped according to the bound unit(s) in which they occur.

- File system structures

- Record locking pool control block
- Volume descriptor blocks (VDBs)
- Directory descriptor blocks (DDBs)
- File descriptor blocks (FDBs)
- Currency control blocks
- Remote extent blocks
- Wait control blocks
- User control blocks
- Semaphore control blocks
- Record locking control blocks
- Device descriptor blocks (DDBs)
- Buffer control blocks
- Public buffer pool headers (BPHs)
- Buffer control blocks (BCBs)
- Buffers.

Each block is assigned an integer that corresponds to the level of the block in the hierarchy. The headings of all blocks are indented according to the depth of the block. This makes it easy to see which files belong to volume major directories and which belong to subordinate directories. The display of the tree of file system structures may be suppressed by the -NF argument.

The following file system structures are also displayed:

- Free indirect request block queue (only when editing a dump file)
- Globally sharable bound units
  - Bound unit description
  - Bound unit attributes
  - Bound unit.

#### TASK RELATED INFORMATION

The preceding logical dump information is obtained from the operating system area of memory and occurs once within a logical dump. The following information can be repeated more than once depending on the number of active pools, task groups, and tasks. This information is presented in the following order:

1. Memory pools (as allocated at CLM time), if there are task groups assigned to them
2. Task groups within a memory pool
3. Tasks within a task group.

## Memory Pool Structures

The following information on sharable bound units is repeated for each pool with assigned task groups:

- Bound unit description
- Bound unit attributes
- Bound unit.

## Task Group Structures

The following information is repeated for each task group in a pool.

- Edited task group information
  - User name, account, and mode
  - Assigned memory pool
  - Bit map switches
  - Outstanding requests to system group
  - Address and name of control block for current working directory, error-out, and user-out.
- Group control block
- Logical resource table
- Logical file table
- Task structures (detailed below)
- File control blocks (if there are active files)
- Work space blocks.

### NOTES

1. For the system task group, IRBs (and hence also RBs) are displayed only when DPEDIT is processing a dump file; i.e., the display is suppressed when the input is from current main memory.
2. Work space blocks and FCBs for the batch task group are not displayed when the batch group is rolled out.

## Task Structures

The following information is repeated for each task in a group:

- Edited task information
  - Bound unit name, location, and start address
  - Hardware level
  - Logical resource number
  - Enabled trap bit map
  - Reserved and current overlay area locations
  - Control block name and address for user-in and command-in.
- Segment descriptor table (swap pool only)
- Memory control block for each segment (swap pool only)
- Task control block
- Trap save area
- MCL word space (for an MCL trap)
- Bound unit description
- Bound unit attributes
- Bound unit
- Overlay areas (if an overlay area table was used).

The firmware-defined fields (instruction, P-counter, I', Z, A, R3, and B3) for each trap save area (TSA) are displayed. If the instruction is a monitor call, the function code is also displayed.

In addition, a possible context of the remaining data and address registers (R1, R2, R4, R5, R6, R7, B1, B2, B4, B5, B6, and B7) is displayed for each trap save area. This context, which is extracted from the work space area of the trap save area, may not be valid in all cases, but in general, is correct due to internal conventions of the Executive.

## DPEDIT Command

The DPEDIT command loads the Dump Edit utility program. Immediately after Dump Edit begins executing, a message is issued to the error-out file giving the unique version number in the following format: DPEDIT-nnn-mm/dd/hhmm. The message "DUMP COMPLETE" is issued to the error-out file immediately before the execution of Dump Edit terminates. The format for the DPEDIT command is:

DPEDIT [path] [ctl\_arg]

path

Pathname of the memory dump file to be printed.

ctl\_arg

None or any of the following control arguments may be entered, in any order:

-NO\_LOGICAL | -NL

Does not print a logical dump of system control structures. Default: Prints a logical dump.

-NO\_PHYSICAL | -NP

Does not print a physical dump of memory. Default: Prints a physical dump.

-NO\_FILES | -NF

Does not print a logical dump of File System structures. Default: Prints File System structures.

-GROUP id [id]... | -GP id [id]...

Produces only group-related information within a logical dump for the group(s) indicated by id; id is the two-character group identifier.

-FROM X'hhhhhhhh' | -FM X'hhhhhhhh'

Low-memory address of area that will appear in physical dump; must be a 1- to 8-character physical address specified in hexadecimal. Default: Absolute 0.

-TO X'hhhhhhhh'

High-memory address of area that will appear in physical dump; must be a 1- to 8-character physical address specified in hexadecimal. Default: High-memory address of the dump file.

**-MEMORY | -MEM**

Produces a dump of main memory. If both the path argument and this argument are specified, an error occurs. Default: Prints a dump of main memory.

**-NO\_SHAREDDBU | -NS**

Does not print sharable and global bound units in the logical dump.

**-NO\_SYS**

Does not print the system portion of the logical dump.

**-ME**

Dump the group that DPEDIT is running in. (This is equivalent to entering: DPEDIT -MEM -NP -NS -NO\_SYS -GROUP my\_group\_id.)

**-FORCE**

Forces DPEDIT to try to edit an incomplete dump file. (i.e., the message "DUMP FILE IS INCOMPLETE" occurred). The results may or may not be useful.

**-SWAP\_FILE swapfile\_name | -PF swapfile\_name**

Specifies the name of a swap file containing non-resident memory information associated with the dump file. Default: No non-resident memory information is available.

**-PSYS**

Dumps only system space in the physical dump.

**-PAGE\_FILE pagefile\_name | -PF pagefile\_name**

Specifies the name of a page file containing non-resident memory information associated with the dump file. Default: No non-resident memory information is available.

**Example 1:**

```
DPEDIT ^DMPVOL>DUMPFIL -NL -TO X'3000'
```

This command loads the Dump Edit utility and requests only a physical dump of the first 12K locations of the specified dump file.

**Example 2:**

```
DPEDIT -MEM -GROUP XX -NP -NF
```



By specifying a group that does not exist (i.e., XX), this command loads the Dump Edit utility and requests an abbreviated logical dump consisting of only the system summary of the currently executing system.

### Operating Procedure for DPEDIT

The DPEDIT utility can be used to examine either the contents of a file created by the previous execution of the MDUMP utility or the contents of the main memory of the system on which DPEDIT is executing. If DPEDIT is being used to examine MDUMP output, mount the disk volume that contains the memory image obtained from the MDUMP memory dump. Once the volume is loaded, specify the disk volume pathname when entering the DPEDIT command.

If DPEDIT is being used to print the contents of either MDUMP output or live memory, the DPEDIT command must be entered with the proper control arguments.

DPEDIT processing can be stopped at any time by pressing the BREAK key. A **\*\*BREAK\*\*** message appears on the user's terminal display when processing stops. An operating system command may be specified at this point. If the Unwind (UW) command is specified, the end-of-processing details are automatically handled and control returns to the command processor with a successful subtask completion status. If the Start (SR) command is specified, DPEDIT resumes processing. If DPEDIT appears to be looping, the loop can usually be broken and DPEDIT can be made to recover by forcing a **\*\*BREAK\*\*** and entering the Program Interrupt (PI) command. Note, however, that it is normal for DPEDIT to run for five or ten minutes (or more) while dumping a large memory or dump file.

### INTERPRETING AND USING MEMORY DUMPS

This subsection describes significant locations in memory dumps, how to interpret the contents of locations on memory dumps, and how to use memory dumps to perform the following procedures:

- Finding the location in memory of your code
- Determining the execution state of your code
- Determining where a trap occurred.

A trap is a special software- or hardware-related condition that may occur during the execution of a task. Many traps are caused by an error, but a few, such as the Monitor Call, are not. The above procedures may have to be performed if a trap message is issued.

Table 9-1 describes memory locations on the dump that may be useful to refer to during debugging. It is assumed that you are familiar with the data structures referenced. Brief definitions of these data structures are contained in the glossary of the System Concepts manual. The locations listed in Table 9-1 are for a single CPU system. If your system has more than one CPU, the data structures are repeated in locations 0100 through 01FF for the second CPU, locations 0200 through 02FF for the third CPU, and so on.

To locate your code, use the logical dump format and locate your group-id and the TCB for your bound unit (BU). The first six characters of the BU filename are printed beside each TCB of the group in a logical dump.

Table 9-1. Significant Locations on Memory Dump

| Memory Address | Meaning  |
|----------------|--|
| 0010/0011      | Head of queue of available trap save areas (TSAs).   |
| 0018/0019      | Pointer to system control block (SCB). This is the key to locating all system data structures.   |
| 0020-0023      | Level activity flags for levels 0 through 63. Bits ON indicate which levels are ready to execute. The lowest (numerically) of these levels is the level currently executing (i.e., the active level). The level 63 bit always is on. The clock level bit (4) may be on.  |
| 0024-007F      | Trap vectors. Each trap vector is associated with a specific trap condition and points to that trap handler's entry address. The trap vector for trap number 1 is in location 7E/7F. The trap vectors for subsequent trap numbers are in descending, contiguous locations; i.e., the trap vector for trap number 2 is in location 007C/007D. |
| 0080-00FF      | Pointers to interrupt save areas (ISAs) for levels 0 through 63, respectively. A null value means there is no dedicated task (i.e., driver) or nondedicated task ready to execute on the specified level.  |

## Determining the State of Execution at the Time of the Dump

Dump analysis begins with gathering all relevant information: the dump itself, the console hard-copy (if any) of the activity of a particular group (or groups), copies of the >>SID>CLM\_USER and >>START\_UP.EC files, plus any link maps.

These materials are required to understand the environment of the system represented in the dump.

Three conditions are discussed below:

1. Halt at level 2.
2. User level active at the time of dump.
3. No level active at the time of dump, except level 63.

### HALT AT LEVEL 2

Examination of the level activity indicators at locations 20-23 confirms that level 2 is active. The system forces this condition to occur if either TSA or IRB resources are exhausted (see CLM SYS directive in the System Building and Administration manual). Note that once level 2 becomes active, other lesser priority levels may be ready but will not receive CPU time.

The D1 register contains an ASCII "IR" (4952) when IRB exhaustion has occurred. Location 10/11 is zero when TSA exhaustion has occurred.

If this symptom persists after augmenting the number of TSA/IRBs available to the system, it is possible that either your code or the system is improperly altering the TSA/IRB chains.

To verify this, take a memory dump immediately after system startup. This allows easy location of the TSA chains from location 10/11 and the IRB chains from the first location of the SCB. Compare this dump to one taken after all TSA/IRBs are supposedly exhausted to verify that they really are. If the system is suspect, supply both dumps to Honeywell. TSAs can also be exhausted by a recursive trap. A recursive trap uses up all available TSAs. Adding TSAs simply allows for greater recursion. In this instance, the system is suspect and dumps should be supplied to Honeywell.

The optionally configured defective-memory trap handler may also force a level 2 halt if a defective memory trap indicates the operating system's trap save area is exhausted. In this case, \$D1 (\$R1) contains X'DEFA'; \$B1, the physical address of the defective memory; and \$B2, the logical address of the defective memory.

## USER LEVEL ACTIVE AT THE TIME OF DUMP

This often indicates a halt or software loop condition on the active level. When a level is active, the pointer to the TCB associated with the code running is in the interrupt vector for that level. Match the TCB pointer with the TCBs listed for the groups present in the system. When a level is active, use the P-counter in the ISA portion of the TCB to locate the software running at the last time this level's context was saved. Since the system clock is active on level 4, the P-counter in the ISA for this level is usually helpful. It is also helpful to record the contents of R and B registers and EO when entering STEP mode at the control panel prior to taking the dump.

## NO LEVEL ACTIVE AT THE TIME OF DUMP

This condition usually indicates a system failure in that all tasks have been suspended and none are being reactivated. In this situation it is helpful to determine the conditions existing at this time. To do this, examine all TCBs in groups other than the \$\$ group. If the TCB under examination has not experienced a default trap condition, it may or may not have an associated TSA. If a TSA is shown, DPEDIT displays the monitor call function code if the trapped instruction is 0001 (monitor call generic).

When the system is called for a monitor function, only those registers that must be preserved by the system are saved in the TSA workspace. The saved registers are: B7, B6, B5, B1, R5, R4, M1, beginning at TSA location E/F. The trap save area (TSA) is detailed below.

### Locating a Trap Processed by the System Default Handler

If a trap message occurs on the operator terminal from the system default trap handler, i.e., "(id) BUname (0303zz) level", the TCB of the referenced task group may be located using the bound unit name (BUname). In this situation, unless the TCB is subsequently requested, the last two TSAs associated with the TCB are related to the system handling of the trap. The first TSA following the TCB was used by the system to terminate the task request in progress when the trap occurred. Your information is found in the next TSA associated with the TCB. It contains the previously described hardware information, followed by a complete set of registers current when the trap occurred. The order of the registers, beginning at location E/F of the TSA, is: B7, B6, B5, B4, B2, B1, I, R7, R6, R5, R4, R2, R1, M1 (B3, R3, I are already in the TSA). When the TCB has been rerequested, only this second TSA remains attached to the TCB.

## USING XRAY

The XRAY utility is used to produce an annotated display of memory image information. Displays produced by the XRAY utility are written to the user-out file. The information to be displayed will be in response to interactively entered commands from your terminal, a stream of commands from a command file, or the processing of structures which simulate DPEDIT operation. There are two sources of memory image information:

- Files created by the previous execution of the BTDUMP utility. (All or selected portions of the file can be dumped.)
- Main memory. (Allows you to determine the configuration under which XRAY is executing.)

You can select one of three display modes in which memory image information is displayed:

1. Raw mode, which provides raw data with no annotation.
2. Pack mode, which provides the name, offset, and contents of each field.
3. Full mode, which provides detailed annotation for each field. This mode is recommended for users new to the system or to reading dumps.

### Accessible Structures

The following list describes the memory locations that may be useful to refer to when XRAY is being used to view information interactively during debugging. It is assumed that you are familiar with the data structures referenced. Brief definitions of these data structures are contained in the glossary of the System Concepts manual.

The data structures that are accessible when running XRAY interactively are:

| <u>Structure</u> | <u>Description</u>                         |
|------------------|--|
| AIFCB            | After image file control block.            |
| BAS              | Bound unit attribute section.              |
| BCD              | Buffer control block.                      |
| BIFCB            | Before image control block.                |
| BPA              | Buffer pool array descriptor.              |
| BPCB             | Buffer pool control block.                 |
| BPH              | Buffer pool header.                        |
| BPSTAT           | Buffer pool status block.                  |
| BUD              | Bound unit descriptor.                     |
| CB               | Control block (record locking structures). |
| CCB              | Currency control block.                    |

| <u>Structure</u> | <u>Description</u>                      |
|------------------|---|
| CQB              | Communications queue block.             |
| FCB              | File control block.                     |
| FDB              | File descriptor block.                  |
| FIB              | File information block.                 |
| FIRB             | Free indirect request block.            |
| GCB              | Group control block.                    |
| IORB             | Input/Output request block.             |
| IRB              | Indirect request block.                 |
| LB               | Lock block (record locking structures). |
| LUD              | Load unit descriptor.                   |
| LFT              | Logical file table.                     |
| LRT              | Logical resource table.                 |
| MCB              | Memory control block.                   |
| MPA              | Memory pool array.                      |
| MPD              | Memory pool descriptor.                 |
| OAT              | Overlay area table.                     |
| OATS             | Overlay area information..              |
| RCT              | Resource control table.                 |
| RXB              | Remote extent control block.            |
| SCB              | System control block.                   |
| SDT              | Segment descriptor table.               |
| SYMBT            | Symbol table descriptor.                |
| TCB              | Task control block.                     |
| TSA              | Trap save area.                         |
| UCB              | User control block.                     |
| WB               | Wait block (record locking structures). |
| ZQCl             | Channel table descriptor.               |
| ZQCDIR           | Channel table directory.                |
| ZQCIT            | Communications information table.       |
| ZQSl             | Station table descriptor.               |

### XRAY Command

The XRAY command loads the Interactive Memory Dump Editor utility program. The format for the XRAY command is:

```
XRAY [path] [ctl_arg]
```

path

Pathname of the memory dump file to be processed.

ctl\_arg

None or any of the following control arguments may be entered:

```
-CUSTOM_STRUCT path | -CS path
```

Specifies the pathname of a file containing additional or customized structures to be included for the session.  
Default: Use only the provided system structures file.

-IN path

Specifies the pathname of a file containing interactive mode commands. If the command file does not end with a QUIT command, additional commands are expected from the terminal user-in file. Default: Use terminal user-in file.

-FILE\_OUT path | -FO path

Specifies the pathname of a file that will receive the printable output. Default: Use terminal error-out file.

-OUT path

Specifies the pathname of a file to which XRAY copies the interactive commands if the commands execute successfully.

-FORCE

Forces XRAY to try to edit an incomplete dump file. (i.e., the message "DUMP FILE IS INCOMPLETE" occurred). The results may or may not be useful.

-SWAP\_FILE swapfile\_name | -SF swapfile\_name

Specifies the name of a swap file containing non-resident memory information associated with the dump file. Default: No non-resident memory information is available.

-DPEDIT

Request Dump Edit simulation. XRAY is placed in non-interactive mode and output is directed to the user-out file. At the termination of Dump Edit simulation, XRAY terminates. Default: XRAY runs in interactive mode.

The following control arguments relate to Dump Edit simulation and must be preceded by a -DPEDIT control argument. This is the only restriction on the order of XRAY control arguments:

-NO\_LOGICAL | -NL

Does not print a logical dump of system control structures. Default: Prints a logical dump.

-NO\_PHYSICAL | -NP

Does not print a physical dump of memory. Default: Prints a physical dump.

**-NO\_FILES | -NF**

Does not print a logical dump of File System structures.  
Default: Prints File System structures.

**-GROUP id [id]... | -GP id [id]...**

Produces only group-related information within a logical dump for the group(s) indicated by id; id is the two-character group identifier.

**-FROM hhhhhhhh | -FM hhhhhhhh**

Low-memory address of area that will appear in physical dump; must be a 1- to 8-character physical address specified in hexadecimal. Default: Absolute 0.

**-TO hhhhhhhh**

High-memory address of area that will appear in physical dump; must be a 1- to 8-character physical address specified in hexadecimal. Default: High-memory address of the dump file.

**-NO\_SHAREDBU | -NS**

Does not print sharable and global bound units in the logical dump.

**-NO\_SYS**

Does not print the system portion of the logical dump.

**-ME**

Dump the group that DPEDIT is running in. (This is equivalent to entering: DPEDIT -MEM -NP -NS -NO\_SYS -GROUP my\_group\_id.)

**-PSYS**

Dumps only system space in the physical dump.

Example 1:

**XRAY**

This command loads the XRAY utility in the interactive mode.



Example 2:

```
XRAY -IN XRAY_C
```

This command loads the XRAY utility in the interactive mode. User commands will be taken from the file named XRAY\_C.

Example 3:

```
XRAY ^DMPVOL>DUMPFIL -DPEDIT -NL -TO 3000
```

This command loads the XRAY utility in the non-interactive mode and requests only a physical dump of the first 12K locations of the specified dump file. DPEDIT simulation mode should only be used as a backup capability to the DPEDIT utility. XRAY has better error recovery when processing corrupted memory dump files.

Operating Procedure for XRAY

The XRAY utility can be used to examine either the contents of a file created by the previous execution of the MDUMP utility or the contents of the main memory of the system on which XRAY is executing. If XRAY is being used to examine MDUMP output, mount the disk volume that contains the memory image obtained from the MDUMP memory dump. Once the volume is loaded, specify the disk volume pathname when entering the XRAY command.

XRAY processing can be stopped at any time by pressing the BREAK key. A \*\*BREAK\*\* message appears on the user's terminal display when processing stops. An operating system command may be specified at this point. If the Unwind (UW) command is specified, the end-of-processing details are automatically handled and control returns to the command processor with a successful subtask completion status. If the Start (SR) command is specified, XRAY resumes processing. If XRAY appears to be looping, the loop can usually be broken and XRAY can be made to recover by forcing a \*\*BREAK\*\* and entering the Program Interrupt (PI) command. Note, however, that it is normal for XRAY to run for five or ten minutes while dumping a large memory or dump file.

If XRAY is being used to print the contents of either MDUMP output or live memory, the -DPEDIT command argument must be entered with the XRAY command with the proper DPEDIT control arguments.

If XRAY is being used to view information interactively, enter the XRAY command (without the -DPEDIT argument) to load the XRAY utility program. You are asked to select one of the following:

- DPEDIT, which allows you to create a memory dump

- QUIT, which allows you to terminate XRAY
- IA, which allows you to examine information interactively.

After you have selected IA, the following online help is available by typing:

- HELP, to get a list of available commands
- HELP followed by any of the commands, to get information about the specific commands and arguments.

The interactive commands are:

```
{ BACKUP }
{ BACK  }
{ B     }
```

This command allows you to step back to the previous structure in the current access path ("Current Path:" display shown in top line of screen display) and display it in zone 1. The structure pointed to by the new last entry of the current access path is accessed and displayed in zone 1.

```
{ CHAIN } fld_name
{ CH    }
```

This command allows you to enter a field name that is appended to the access path and then displayed in zone 1. If the field named by the fld\_name argument is valid (found in the structure currently being displayed) and is a pointer to a structure, the pointed-to structure is accessed and displayed in zone 1. The structure's name is also appended to the current access path.

```
{ CONTINUE }
{ C        }
```

This command allows you to redisplay the current structure in zone 1. The structure pointed to by the last entry in the current access path is accessed and redisplayed in zone 1. This refreshes the data being displayed in zone 1. This is important in controlling which zone's data is current for displaying indicators or single zone (full screen) displays.

CS path\_name

This command allows you to specify a customer structure file containing additional or customized structures that are to be used during the session. The customer structure file named in the XRAY command argument (if any) is overridden.

```
{ DISPLAY }  
{ DISP  
D }
```

This command allows you to redisplay the current structure in zone 2. The currently assigned zone 2 structure is the structure that was accessed by a DISPLAY command with the fld\_name argument. This structure is accessed and redisplayed in zone 2. This refreshes the data being displayed in zone 2. This is important in controlling which zone's data is current for displaying indicators or single zone (full screen) displays.

```
{ DISPLAY } fld_name  
{ DISP  
D }
```

This command allows you to display a field in zone 2. If the field named by the fld\_name argument is valid (found in the structure currently being displayed) and is a pointer to a structure, the pointed to structure is accessed and displayed in zone 2. There is no effect on the current access path.

```
{ DISPLAY } IND  
{ DISP  
D }
```

This command allows you to display indicators in zone 2. The most recently accessed structure, either in zone 1 or 2, is examined for any fields defined as indicator or flag fields and the individual indicators are displayed in zone 2.

```
{ DISPLAY } { FROM } hhhhhhhh [P]  
{ DISP } { FM }  
D }
```

This command allows you to use a virtual or physical address to display the pointed-to data area in zone 2. The address specified by hhhhhhhh is accessed and the contents are displayed in zone 2. If there is no structure associated with the data, the USE command can be used to associate a structure to this data. If the P argument is used, the address is accepted as a physical address, otherwise the address is treated as a virtual address.

DPEDIT [ctl\_arg]...

This command allows you to create a memory dump. XRAY will leave the interactive mode of operation and perform the DPEDIT simulation controlled by any supplied DPEDIT arguments. Refer to the XRAY command description for a description of these arguments.

## FO path\_name

This command allows you to redirect the output (user-out). XRAY will redirect the user-out to the specified file-out pathname.

```
{ HARDWARE } n  
{ HARD  
H }
```

This command allows you to display the hardware-dedicated area of any processor. The hardware-dedicated information for the processor specified by n is displayed in zone 2. The value for n is an integer from 0 to 15.

## HELP [command]

This command allows you to obtain a list of available interactive commands or help information for a named command. If the command argument is specified, help information for the named command is displayed below the command line. If no command is specified, the name of all commands will be displayed. Responding with a Y to the "more?" prompt causes additional help information.

```
{ LOCATE } str_name [[ctl_arg]]  
{ LOC  
L }
```

This command allows you to search for a specific structure and display that structure in zone 1. If the control argument (ctl\_arg) isn't included, the first occurrence of the structure named by str\_name is accessed and displayed in zone 1. Using the control argument allows the search to continue until a structure named by str\_name meeting the test described by ctl\_arg is found or all occurrences of that structure have been tested.

The control argument must be bounded by brackets ([]) and will be one or more tests comparing A to B. The format is:

[A::B...]

where A can be a field name within the structure or an offset into the structure (beginning with 0). B can be a field name within the structure, a hexadecimal constant, or an ASCII character string constant. The comparison (::) can be:

- > Greater than
- > Less than
- = Equal to
- # Not equal to

- + Bit true
- Bit false

For two or more tests, the individual tests are separated by:

- / Indicating and test results
- : Indicating or test results

For structures containing identifier fields such as G NAME in a GCB structure, the test can be formatted as:

[=id]

where id would be the two-character group id.

```
{ MODE } zone { R }
{ M   }       { P }
                { F }
```

This command can be used to select one of three display modes for either of the two display zones. The zone specified by zone (either 1 or 2) is changed to the mode specified by R, P, or F. The modes are:

1. R for raw mode, which provides raw data with no annotation.
2. P for pack mode, which provides the names, offset, and contents of each field.
3. F for full mode, which provides a detailed annotation for each field.

```
{ NEXT }
{ N     }
```

This command allows you to step to the next version of the structure or element in the current access path and display it in zone 1. If the current structure in zone 1 is one of a queue of like structures or a table of elements, the next structure or element will be accessed and displayed in zone 1.

```
{ PAGEDOWN } zone
{ PDN      }
{ PD       }
```

This command allows you to move a zone display one page of information toward the end of the information. If the structure in the zone specified by zone (either 1 or 2) has more information than is displayed on the screen, this command will cause the next page (screen) toward the end of the structure to be displayed.

{ PAGEUP } zone  
{ PUP }  
{ PU }

This command allows you to move a zone display one page of information toward the beginning of the information. If the structure in the zone specified by zone (either 1 or 2) has more information than is displayed on the screen, this command will cause the next page (screen) toward the beginning of the structure to be displayed.

{ PRINT }  
{ PR }  
{ P }

This command allows you to print the current screen image. The screen image is sent to user-out.

{ PRINT } fld\_name [mode]  
{ PR }  
{ P }

This command allows you to print the contents of a structure. If the field named by the fld\_name argument is valid (found in the structure currently being displayed) and is a pointer to a structure, the pointed to structure is accessed and sent to user-out. The information is printed in raw mode, unless otherwise specified by the mode argument.

{ PRINT } { FROM } hhhhhhh { LIMIT } hhhhhh [P]  
{ PR } { FM } { LM }  
{ P }

This command allows you to use a virtual or physical address to print the pointed-to data area. The address specified by hhhhhhhh is accessed and the contents are sent to user-out in raw mode until the limit number of words have been sent. If the P argument is used, the address is accepted as a physical address, otherwise the address is treated as a virtual address.

{ PRINT } { AUTO }  
{ PR } { MAN }  
{ P }

This command allows you to control when the current screen image will be printed. AUTO causes the screen image to be sent to user-out whenever subsequent commands cause the screen to change. MAN causes the screen image to be printed only on demand.

```
{ QUIT }  
{ QT }  
{ Q }
```

This command allows you to end the current interactive session of XRAY.

```
{ RESTART }  
{ RES }
```

This command allows you to cancel the current access path and start over by displaying the System Control Block. Returns XRAY to first screen displaying HW and SCB in zones 1 and 2.

```
{ SCROLLDOWN } zone n  
{ SDN }  
{ SD }
```

This command allows you to move a zone display a specified number of lines of information toward the end of the information. If the structure in the zone specified by zone (either 1 or 2) has more information than is displayed on the screen, this command causes the next specified n number of lines toward the end of the structure to be displayed.

```
{ SCROLLUP } zone n  
{ SUP }  
{ SU }
```

This command allows you to move a zone display a specified number of lines of information toward the beginning of the information. If the structure in the zone specified by zone (either 1 or 2) has more information than is displayed on the screen, this command causes the next specified n number of lines toward the beginning of the structure to be displayed.

SF swapfile\_name

This command allows you to specify a swapfile that is associated with the dump file being processed. The swapfile named in the XRAY command argument (if any) is overridden.

```
{ USE } str_name zone  
{ U }
```

This command allows you to associate a structure to the displayed data in a zone. Associate the structure named by str\_name to the data that is currently displayed in the zone specified by zone (either 1 or 2).

{ WALK } str\_name  
{ W }

This command allows you to locate and display in zone 1 all occurrences of a structure. As each occurrence of the structure named by str\_name is displayed, you are prompted to either end the walk or continue to the next like structure.

{ ZONES } n  
{ ZONE }  
{ Z }

This command allows you to change the display to either one (full screen) or two zones. A selection of 2 changes the screen image format to display both zones. Selecting 1 changes the screen format to display only the most recently displayed zone.



REMOVE THIS PAGE AND PLACE TAB FOR

TAB 10

PATCH UTILITY



## *Section 10*

# **PATCH UTILITY**

This section describes how to use the Patch utility.

### OVERVIEW

The Patch utility is used to apply patches to and remove patches from object units (variable sequential files created by the compilers) and bound units (relative files created by the Linker). Patches are identified by patch-ids. The Patch utility can also be used to list, by patch-id or group-id, all patches for an object unit or bound unit. The listing is written to the user-out file, terminal line screen, or printer for a hard copy.

Unless you specify otherwise, the patcher does not patch a location that has been patched before.

The Patch utility, in modifying object or bound units, extends the file space, as necessary. Insufficient file space terminates Patch operations; therefore, you should ensure that sufficient space exists to accommodate the patches on the medium (disk, etc.).

## OPERATION

Patch execution is controlled by directives entered to Patch through the terminal user-in file or a sequential file. Only the file specified on the Patch command line can be patched with each invocation of the Patch utility. The Patch utility operates in either absentee mode (batch mode) or in interactive mode. By using Patch directives, you can:

- Manipulate shared and system attributes of bound units
- Assign an address or value to undefined external references in bound units
- Interrogate the current contents of bound unit locations
- Apply patches with or without verifying the existing values at the locations to be patched
- List patches
- Eliminate patches.

The Patch utility performs a verify unique address operation as it applies new patches. That is, as each new patch is applied, it is compared against those already applied. If the new patch modifies a location already modified by a previous patch, the new patch will be rejected and an error message will be issued. If the Patch utility is invoked using the -FORCE argument, the address is not verified and patches to patches will be accepted.

The Patch utility also maintains a special history record for bound units that are distributed by Honeywell. This record contains information pertaining to the update level of the bound unit and the number of patches applied to the bound unit for each update. You can determine from this information if a program is at the correct update level, has the correct number of patches or has had some patches added or deleted. This information can be displayed using either the LS, LP, or LU directives.

### Absentee Mode

In absentee mode Patch processes directives and applies them to the bound unit or object unit file specified on the Patch command line. These directives allow applying of patches with or without verification, elimination of patches, and listing of patches.

The Patch utility processes attribute modification and interrogate directives as they are entered. Regardless of the input sequence of other directives, Patch processes directives in the order: resolve undefined references, eliminate patches, apply patches, and list patches.

## Interactive Mode

By specifying the Patch command with the -IA argument, a bound unit file can be patched in interactive mode. In interactive mode, directives must be completed before they are applied; a directive is completed when the Patch utility reads a new directive.

Manipulation of the bound unit share or system attributes and bound unit interrogation are always performed as the directives are keyed-in.

## Loading Patch

Patch can operate on two types of files:

- Object units, which are variable sequential files created by the compilers
- Bound units, which are relative files created by the Linker.

To load Patch, enter PATCH, as follows:

FORMAT:

```
PATCH filenm [ctl_arg]
```

ARGUMENTS:

filenm

Pathname of the object unit file or bound unit file to be patched. If an object unit is being patched, the pathname must end with the .O suffix.

ctl\_arg

The following control arguments can be entered:

-IA

Operate in interactive mode. Process one directive at a time as they are entered; error messages (if any) immediately follow the applicable directive. If this argument is not specified, Patch operates in the absentee mode. Object unit files must be patched in absentee mode.

**-IN path**

Pathname of the device through which Patch directives are entered; can be your terminal, a card reader, or a sequential file. Error messages are written to the error-out file. Patch error messages are described in the System Messages manual. Default: User\_in.

**-PROMPT | -PT**

If input is from your terminal, each time the Patch utility program is ready to accept an input line, the typeout P? appears on the input device. Default: No prompt.

**-SI**

Suppress the display of the sign-on message (i.e., PATCH, followed by the system version number and the date Patch was created). Default: Patch sign-on message is displayed.

**-SIZE nn | -SZ nn**

Create a Patch work area of nn 1024-word blocks of memory. "nn" specifies the maximum number of blocks and must be from 01 to 63. Default for nn: 10.

**-FORCE**

Suppress the verify unique address operation as new patches are applied. Normally, if the new patch modifies a location already modified by a previous patch, the new patch will be rejected and an error message will be issued. If this argument is used, the address is not verified and patches to patches will be accepted. Default: perform the verify unique address operation.

Submitting Patch Directives

The Patch directives are listed and briefly defined below. Detailed descriptions for each Patch directive are provided later in this section.

| <u>Directive Name</u> | <u>Function</u>  |
|-----------------------|--|
| CLSY                  | Set bound unit system bit off  |
| DP                    | Apply patches to either the data section of a bound unit or to the common area of an object file |
| EP                    | Eliminate named patch or group of patches, or all patches  |

| <u>Directive Name</u> | <u>Function</u>  |
|-----------------------|--|
| GO                    | Process previous patch directive if mode is interactive  |
| GP                    | Apply a group (logical set) of patches to a bound unit   |
| GNSH                  | Set bound unit global share bit off  |
| GSHR                  | Set bound unit global share and share bits on  |
| HP                    | Apply hexadecimal patches  |
| LDEF                  | Assign an address to an undefined external location reference within a bound unit  |
| LG                    | List groups (logical sets) of patches by name only or specified groups within a bound unit and exit from Patch if mode is absentee (batch) |
| LN                    | List patches now but do not exit from Patch if mode is absentee (batch)  |
| LP                    | List patches and exit from Patch if mode is absentee (batch)   |
| LS                    | List patches by name only or specified patch and exit from Patch if mode is absentee (batch)   |
| LU                    | List the Honeywell RSUF updates that were applied to the bound unit  |
| NS                    | Set bound unit global share and share bits off   |
| Q or QT               | Process previous Patch directives if mode is absentee (batch) and exit from Patch  |
| SD                    | Apply symbolic patches to either the data section of a bound unit or to the common area of an object file                                  |
| SP                    | Apply symbolic patches   |
| SS                    | Set bound unit share bit on  |
| STSY                  | Set bound unit system bit on   |
| VDEF                  | Assign a value to an undefined external symbol within a bound unit   |
| WA                    | Interrogate bound unit locations   |
| *                     | List a comment on the user-out file  |

Each Patch directive consists of only a directive name or a directive name followed by one or more values. Values must be separated by a delimiter. The delimiter can be a space, a comma, or a semicolon. However, on an interactive device (i.e., a terminal), the carriage return replaces the delimiter. Lines can neither begin nor end with a comma or semicolon. If directives are entered from a card reader, trailing blanks or column 80 replace the delimiter.

Multiple Patch directives can be specified during one execution of the Patch utility. To enter Patch directives for a different file, you must reload Patch, specifying a different file in the filename argument.

For patching in the interactive mode:

- Patch directives are processed in the sequence in which they are entered.
- Patch directives can be entered in any order, except that Quit (Q) must be entered last.
- A Patch directive must be complete before it is processed; it is complete when Patch reads a new directive.

For patching in the absentee (batch) mode:

- The List Patches Now (LN) directive must be the first directive; otherwise, it is processed like an LP directive (i.e., last).
- Patches are first eliminated, then applied, and finally listed regardless of the sequence in which the associated directives are entered.
- The bound unit share bit and system bit (SS, STSY, CLSY, GSHR, GNSH, and NS) directives and the Interrogate bound unit (WA) directive are always processed when they are entered.

If directives are being entered through a terminal, press RETURN at the end of each line. Each time RETURN is pressed, except after Quit, the typeout P? is reissued if the prompt control argument was specified in the command line.

Use the BREAK key with caution; it can only be followed by the SR command when applying or eliminating patches. BREAK followed by the UW, PI, or NEW\_PROC command while applying or eliminating patches produces a corrupted file. BREAK can be used safely with UW, PI, or NEW\_PROC only if listing patches or between directives in interactive mode.



## PATCHING TECHNIQUES

Techniques used when naming and applying a Patch are described in the following paragraphs.

### Naming the Patch

Each patch has a patch-id by which it is identified. When you designate in Patch directives (DP, HP, SD, or SP) that one or more patches are to be applied to a specified object unit or bound unit, you must specify a patch-id. The patch-id identifies the patch(es) and designates whether the patch(es) are to be applied to an object unit or to the root, data section, or overlay of a bound unit.

To eliminate individual patches from or list individual patches in an object unit or bound unit, you must specify in the directive the patch-id with which the patch(es) are associated. See "Hexadecimal Patch (HP)" for a description on how to designate patch-ids.

### Applying the Patch

If an object unit is being patched, object records are created for the specified patches and appended to the end of the object file. These records are referred to by the LS, LP, LG, and EP directives. When the object unit is processed by the Linker, existing values are replaced with the specified patch values. Locations that contain external references should not be patched.

If a bound unit is being patched, each specified patch value is applied directly to the proper image record in the bound unit. The patch-id, the previous value, and the patch value are saved in a Patch history record that is written at the end of the file area allocated to the bound unit. These records are referred to each time a List Patch or Eliminate Patch directive is specified.

Use caution when patching executing bound units. If a program or one of its overlays is loaded while in the process of being patched, results are unspecified.

## PATCH DIRECTIVES

The Patch directives are described on the following pages in alphabetic order by directive function.

# CLEAR SYSTEM BIT

## CLEAR SYSTEM BIT (CLSY)

Indicate that the patched bound unit is prohibited from running as a system task. This directive turns off an indicator in the bound unit header area. This directive cannot be used for object unit files.

The system bit is set at link time by the SYS Linker directive.

FORMAT:

CLSY

Example:

CLSY

In this example, the bound unit header contains an indicator signifying that the bound unit is prohibited from running as a system task.

## COMMENT

### COMMENT (\*)

List the accompanying text on the user-out file. The contents of the Comment directive are not saved in the patch history file. Permanent comments can be entered with symbolic patch directives. For an example of this method, refer to the Symbolic Patch (SP) description found later in this section.

#### FORMAT:

\* comment-text

#### Example:

\* THIS IS A COMMENT

In this example, the phrase THIS IS A COMMENT is displayed when this directive is executed.

# DATA PATCH

## DATA PATCH (DP)

For bound units, apply one or more hexadecimal patches, by relative location, to the data section of the bound unit. The bound unit must have been created by the Linker when the -R Linker argument is specified (separate code and data section).

For object files, the DP directive causes patches to be applied to common areas.

### FORMAT:

For Bound Units, Without Verification:

```
DP patch-id /addr patchval[ patchval...][ /addr patchval...]
```

For Bound Units, With Verification:

```
DP patch-id /addr (verval patchval[ verval patchval...])  
[/addr (verval patchval[ verval patchval...])...]
```

For Object Files, Without Verification -- Local Common Block:

```
DP patch-id /offset1 patchval[ /offset1 patchval]...
```

For Object Files, With Verification -- Local Common Block:

```
DP patch-id /offset1 (verval patchval)  
[/offset1 (verval patchval)]...
```

For Object Files, Without Verification -- Named Common Block:

```
DP patch-id blockname /offset2 patchval[ patchval...]  
[/offset2 patchval[ patchval...]]...
```

For Object Files, With Verification -- Named Common Block:

```
DP patch-id blockname /offset2 (verval patchval  
[ verval patchval]...)[/offset2 (verval patchval[ verval  
patchval]...)]...
```

## ARGUMENTS:

## patch-id

Patch-id of the patch(es) to be applied. A patch-id comprises eight characters; the first six can be any ASCII characters except spaces. The last two characters must be RT for an object unit or the separate data area of a bound unit linked with the -R option.

## /addr

Relative location in the bound unit segment at which the first (or only) subsequent patch value is applied. Each address must comprise one to eight right-justified hexadecimal characters and must be preceded by the slash character (/). Subsequent patch values, if any, are applied to succeeding memory locations. A patch can have a maximum of 127 /addr fields and a maximum of 127 values for any /addr field. The segment base is added to all locations specified or implied and all relocatable addresses (IMAs).

## NOTE

Take care in specifying an address to be patched. If the address of a location to be patched is identified when a bound unit is being executed, that memory address has three possible factors:

- The original address of the location in the bound unit relative to the beginning of the bound unit
- The linking relocation factor
- The loader relocation factor.

If a bound unit address that is to be identified at execution time is being patched, the loader relocation must be subtracted from the address. If an object unit is being patched, both the linking and loader relocation must be subtracted. Object unit locations can also be obtained from the listing produced during assembly.

## DATA PATCH

### patchval

Value to be inserted at an address, replacing the contents of that location. The value must be specified as one of the following:

- Data, represented by one to four hexadecimal characters
- Relocatable address, represented by one to eight hexadecimal characters, preceded by the less-than character (<).

### verval

Verification value; one to four hexadecimal characters specifying the value that should be in the location before the patch is applied. If patchval is a relocatable address, verval can be one to eight hexadecimal characters.

### offset1

Non-negative offset from the beginning of \$LCOMW.

### patchval

A value of one to four hexadecimal characters to insert into \$LCOMW. Relocatable values are not permitted and only one patch value can be specified for each offset.

### blockname

Symbolic name of the common block. The name can contain one to six characters. Only one blockname is allowed per directive.

### offset2

Offset from the symbol name of the common block.

### NOTES

1. Each verval must be immediately followed by a patchval.
2. The verification value(s) and patch value(s) associated with each address must be enclosed within parentheses.

3. For consecutive locations, the verify and new values can be included within one set of parentheses. The /addr field is internally adjusted.
4. Within a set of parentheses, the number of verify values must equal the number of new values.
5. The IMA indicator cannot be used with a verify value. IMA status is determined from the segment or from the new value.
6. For IMAs, verify value and new value can be up to eight hexadecimal characters (30 bits). If the new value is not an IMA, the verify value can be no more than four hexadecimal characters even if the old value is an IMA.
7. For IMAs, Patch allocates two words. For example:

DP patch-id,/100,(1111,<12345,ABC,DEF)

If the contents of 100 and 101 are 00001111, and the contents of 102 are 0ABC, the patch is applied, and the contents of the specified addresses are:

| <u>Address</u> | <u>Contents</u> |
|----------------|-----------------|
| 100            | 0001            |
| 101            | 2345            |
| 102            | 0DEF            |

8. An IMA can be patched to a non-IMA or a non-IMA can be patched to an IMA.
9. Verified and nonverified patches can be included within one patch directive; however, if the verify fails, none of the addresses in the directive is patched.
10. A left parenthesis cannot immediately follow a right parenthesis or unverified patch value. There must be a /addr field between them.
11. In object modules, patches to areas that have no defined value cannot be verified.

# ELIMINATE PATCH

## ELIMINATE PATCH (EP)

Eliminate all patches or all patches associated with a specified patch-id or group-id. The patch(es) must have been previously applied by DP, HP, SD, or SP directives. To determine what patches have been applied, and their patch-ids, enter one of the List Patch (LN, LP, LS) directives described later in this section.

If you are eliminating patches to patches, you must specify the patch-ids or group-ids in reverse order of application to preserve the integrity of the bound unit locations involved.

If you eliminate the patches using the ALL argument, or out of reverse order, and the bound unit contains patches to patches, the locations with multiple patches are not restored to their original value. When patcher detects this situation, it issues a warning message, but continues to eliminate the patches as directed.

In absentee mode, the Patcher eliminates specified patch-ids or group-ids in reverse patch history order, no matter in what order the EP directives are entered.

### FORMAT:

EP { patch-id }  
    { group-id }  
    ALL

### ARGUMENTS:

patch-id

Patch-id of the patch(es) to be removed. A patch-id comprises eight to ten characters: the first six can be any ASCII characters except spaces. The last two to four characters must identify the root or overlay to which the patch(es) are applied. If an object unit or the root or separate data area of a bound unit is being patched, the patch-id is eight characters, the last two of which are RT. If an overlay is being patched, the last two to four characters identify the hexadecimal overlay number. The first overlay is 00 for bound units created by the Linker (01 if linked with the -R option), and subsequent overlays are numbered consecutively in ascending order. There can be no embedded blanks. Within the root and each overlay, patch-ids must be unique.



group-id

Group-id of the group of patches to be removed. A group-id comprises six ASCII characters and is assigned to the group by the GP directive. Note that single patches within a group patch cannot be removed - they must be removed as a group.

ALL

If the ALL option is used, all patches in the file are eliminated in the order they were applied.

Example 1:

EP NUMBRFOA

In this example, patch NUMBRF in overlay 0A of a bound unit is eliminated.

Example 2:

EP W9999A

In this example, patch group W9999A with all its associated patch-ids is eliminated from a bound unit.

**GO**

GO

Tell Patch that the previous directive is complete and is to be processed. This directive is effective only in interactive mode for which a new Patch directive signals the completion of the previous one. The GO directive is used in circumstances in which the user would like to have a directive processed before entering any other directive.

**FORMAT:**

**GO**

## GROUP PATCH

### GROUP PATCH (GP)

Apply two or more patches to a bound unit as a logical set (group). In order to eliminate one patch of the group, the entire group must be eliminated. The group will usually consist of patches to the root and various overlays of a bound unit. The group can be a maximum of 30 patch-ids.

Patch directives for the patch-ids listed in the GP directive must immediately follow the GP directive. If an error is detected for any patch within the group, the entire group patch will not be applied.

#### FORMAT:

```
GP group-id,patchid1,patchid2,....,patchidn
```

#### ARGUMENTS:

group-id

Group-id to be assigned to the group of patches. A group-id comprises six ASCII characters.

patch-id

Patch-id of the patch(es) to be within the group. A patch-id comprises eight to ten characters: the first six can be any ASCII characters except spaces. The last two to four characters must identify the root or overlay to which the patch(es) are being applied. If an object unit or the root or separate data area of a bound unit is being patched, the patch-id is eight characters, the last two of which are RT. If an overlay is being patched, the last two to four characters identify the hexadecimal overlay number. The first overlay is 00 for bound units created by the Linker (01 if linked with the -R option), and subsequent overlays are numbered consecutively in ascending order. There can be no embedded blanks. Within the root and each overlay, patch-ids must be unique.

#### Example:

```
GP W9999A,W9999ART,W9999A00,W9999A01
```

In this example, patch W9999ART in the root, W9999A00 in overlay 00, and W9999A01 in overlay 01 of a bound unit are associated to create a group patch named W9999A.

# HEXADECIMAL PATCH

## HEXADECIMAL PATCH (HP)

Apply one or more individual patches, by relative location, to an object unit or bound unit. You can designate that specified patch(es) be applied only if specified location(s) currently contain specified value(s); these are called verification values. Within a single HP directive, verification values can be specified for some or all of the locations. If any of the verification values do not match the values currently at the locations for which verification values were specified, none of the patches specified in the HP directive are applied.

### FORMAT:

Without Verification Values:

```
HP patch-id, [base, ]/addr, patchval[, patchval...patchval]
[[+base, ]/addr, patchval[, patchval...patchval]]...
```

With Verification Values:

```
HP patch-id, [base, ]/addr, (verval, patchval[, verval, patchval]...)
[[+base, ]/addr, (verval, patchval[, verval, patchval]...) ]...
```

### NOTES

1. One or more lines of arguments can be specified. When two or more lines of arguments are entered in an HP directive, the last character on each line must be a valid hexadecimal character or right parenthesis. Individual fields, values, and addresses must not be split between lines. The entry of a Patch directive name (e.g., EP, LP) at the beginning of a line designates the end of the previous Patch directive.
2. A space or semicolon can be used in lieu of a comma as a separator.

## ARGUMENTS:

## patch-id

Patch-id of the patch(es) to be applied. A patch-id comprises eight to ten characters; the first six can be any ASCII characters except spaces. The last two to four characters must identify the root or overlay to which the patch(es) are being applied. If an object unit or the root of a bound unit is being patched, the patch-id is eight characters, the last two of which must be RT. If an overlay is being patched, the last two to four characters identify the hexadecimal overlay number. The first overlay is 00 for bound units created by the Linker (01 if linked with the -R option), and subsequent overlays are numbered consecutively in ascending order. There can be no embedded blanks. Within the root and each overlay, patch-ids must be unique.

## base

Optional argument allowed only for bound units. Base defines a value that is added to all locations, i.e., /addr specified or implied and all relocatable addresses (IMAs). If this argument is omitted, the default base is the segment base. Base can be entered as a hexadecimal address of one to eight characters or as a name that has been specified as an EDEF at link time and placed in the bound unit symbol table. If a symbol name is used, Patch finds the name in the symbol table and uses its address as the base value. The format for the symbol name as a base is +symname, where symname comprises 1 to 12 characters. Except in the case of multiple base fields, if a hexadecimal address is used for base, the plus sign is not required. Leading hexadecimal digits A through F should be preceded by 0 to assure treatment as a numeric base.

For bound units created by the Linker, the values specified for the /addr fields and IMA references (if any) must include the displacement of the root or overlay. This displacement is equal to the base address of the root or overlay as printed on the link map. The user can specify the base argument in the Patch directive, omit it and rely on the default base value, or specify a zero base and add the displacement to each /addr field and IMA to achieve the same result. For an example of all methods, refer to "Symbolic Patch (SP)" later in this section.

## HEXADECIMAL PATCH

/addr

Relative location at which the first (or only) subsequent patch value is applied. Each address must comprise one to eight right-justified hexadecimal characters and must be preceded by the slash character (/). Subsequent patch values, if any, are applied to succeeding memory locations. A patch can have a maximum of 127 /addr fields and a maximum of 127 values for any /addr field.

### NOTE

Take care in specifying an address to be patched. If the address of a location to be patched is identified when a bound unit is being executed, that memory address has three possible factors:

- The original address of the location in the bound unit relative to the beginning of the bound unit
- The linking relocation factor
- The loader relocation factor.

If a bound unit address that is to be identified at execution time is being patched, the loader relocation must be subtracted from the address. If an object unit is being patched, both the linking and loader relocation must be subtracted. Object unit locations can also be obtained from the listing produced during assembly.

patchval

Value to be inserted at an address, replacing the contents of that location. The value must be specified as one of the following:

- Data, represented by one to four hexadecimal characters
- Relocatable address, represented by one to eight hexadecimal characters, preceded by the less-than character (<).

verval

Verification value; one to four hexadecimal characters (one to eight if patchval is a relocatable address) specifying the value that currently should be in the location at which the subsequent patch will be applied. See the notes on verification that follow the DP directive.

Example 1:

```
HP PTCHIDRT,/1B2A,1FFF,1DFC,<2BFC,2D4E,<ABF2
```

This Hexadecimal Patch (HP) directive requests that the subsequent patches, identified by the name PTCHIDRT, be applied to the root. Patch values 1FFF through <ABF2 are to be inserted in successive locations, with the first patch value 1FFF to be located at address 1B2A. The hexadecimal patches are to replace any previous values in these locations. The value to be inserted in address 1B2C is the two word address 2BFC, which is to be relocated at load time; the relocatable address ABF2 is to be inserted in address 1B2F. Note that patch locations and relocatable address values are relative to the root base.

Example 2:

```
HP VPATCH01,/1FEA,(1A1,1B7,1A7,1B8),/1E72,8900
```

This example illustrates the use of verification values in a Hexadecimal Patch (HP) directive requesting that specified patches, identified by the name VPATCH01, be applied to overlay 01. Patch checks location 1FEA for the value 1A1, and location 1FEB for the value 1A7. If the values are at those locations, then the contents of locations are changed as follows: location 1FEA contains 1B7, location 1FEB contains 1B8, and location 1E72 contains 8900. If either of the verification values is incorrect, none of the three locations is changed. Note that patch locations are relative to the overlay base.

# INTERROGATE BOUND UNIT

## INTERROGATE BOUND UNIT (WA)

Display the current contents of specified locations within a bound unit on the user-out file. This directive cannot be used to display locations in object files.

### FORMAT:

WA [ovly,]/addr1[,words][,/addr2...]

### ARGUMENTS:

ovly

Hexadecimal overlay number that the address references. If this field is omitted, the root is the default. The root can also be specified as RT. For -R linked bound units (separated data and code), this field can be CM or DP for data section or RT for code section as well as being an overlay number.

addr

The relative location within the specified root, data section, or overlay indicating where the display is to start.

words

The hexadecimal number of consecutive words to be displayed. The default is one. Eight locations per line are displayed.

Example:

WA 18,/C,2

In this example, two words from the bound unit overlay 18 at offset C are displayed. The format of the display is:

OVLV ADDRESS LOC[ LOC...]

Therefore, the above example would display:

0018 000000C xxxx yyyy

where xxxx is in location C and yyyy is in location D of overlay 18.



LDEF

Assign a specified address to an undefined external location reference and change all locations that reference this name within a bound unit. Undefined external references in a bound unit can only be changed once. If you make a mistake, you must use HP or SP directives to correct each location containing the wrong information. This directive cannot be used for object unit files.

## FORMAT:

```
LDEF symname, [<]addr[,L]
```

## ARGUMENTS:

symname

Name of the undefined external reference that is assigned an address; can be from 1 to 12 characters long. If symname is used as both a relocatable and displacement address, two separate LDEF directives are required.

addr

Address to which symname is assigned.

&lt;

Address specified is a relocatable (IMA) address. If this argument is not used, the address is treated as a displacement ( $P+D\overline{SP}$ ).

L

List all changed external references to symname on the device specified as user-out. Default: No list. No history of the changes made with the LDEF directive is kept. Therefore, use the L argument and retain the listing for future use.

## Example 1:

```
LDEF EPPTR,50,L
```

This directive assigns address 50 to symbol EPPTR and lists all changed locations.

LDEF

Example 2:

LDEF PK,<50,L

This directive assigns address 50 to symbol PK and changes all IMA references to external symbol PK to address 50.

## LIST GROUP PATCH NAMES

### LIST GROUP PATCH NAMES (LG)

List the names (group-ids) of the group patches along with their respective patch names (patch-ids) in the bound unit and exit from Patch if in absentee (batch) mode. The listing is produced on the user-out file. Addresses, values, and symbolics are not listed.

FORMAT:

LG

Example:

LG

Assuming a bound unit is being patched, this example produces a printout of all the names of group patches applied to the bound unit with their respective patch-ids.

The printout would appear as:

```
          W9999A
RT      *W9999A
0000    *W9999A
```

The printout has the following meaning: a group patch identified by group-id W9999A was applied to the bound unit. The group consisted of the patch identified by patch-id W9999ART applied to the root and the patch identified by patch-id W9999A00 applied to overlay 00.

# LIST SPECIFIED GROUP PATCH

## LIST SPECIFIED GROUP PATCH (LG)

List patches for patch-ids associated with those group-ids specified and exit from Patch if in absentee (batch) mode. The listing is produced on the user-out file. Up to five group-ids can be requested per absentee run.

### FORMAT:

LG group-id[,group-id...]

### ARGUMENTS:

group-id

Group-id assigned to the group of patches by the GP directive. A group-id comprises six ASCII characters.

### Example:

LG W9999A

Assuming bound unit patches are being listed, this example produces a printout of the entire group patch W9999A with all its associated patch-ids.

# LIST PATCHES

## LIST PATCHES (LP)

Produce a listing of all patches within the object unit or bound unit being patched and exit from Patch if in absentee (batch) mode. The listing is produced on the user-out file.

If bound unit patches are being listed, the listing designates, for each patch, the following information in the order listed: full patch-id, Honeywell update number if any, address at which the patch was applied, contents of the location before the patch was applied, and the patch value. Listings of patches include any symbolic instructions and comments entered by the user on a Symbolic Patch (SP) directive. A final summary line provides the count of all patches on the file.

If patches on a bound unit that is distributed by Honeywell are being listed, the listing may also include update summary information. The information consists of the current update level of the bound unit and the total number of update-applied patches. A warning is issued if the total number of patches entered through updates is not equal to the actual count of patches on the bound unit.

In the listing, the characters that identify the root or overlay segment appear first, and are separated from the other six characters of the patch-id by spaces. When the separate data area (common) of a bound unit has been patched, the letters CM are printed rather than RT.

When listing a group patch, a group-id lists with blanks in the segment identifier field. An asterisk to the left of the following patch-ids indicate their association with the group.

If object unit patches are being listed, the listing designates the following information for each patch: six character patch-id (omitting RT), address at which the patch was applied, patch value, and any symbolic instruction and comment if present.

If termination of the listing of patches is desired before normal completion of the list process, use the Break facility followed by a UW command. The Patch program must then be reloaded.

FORMAT:

LP

## LIST PATCHES

### Example 1:

LP

Assuming a bound unit file, this example produces a printout of a listing of patches applied to the bound unit:

```
0001 G5195A 02 008C02E2 00000000 00000F02 NOP >$+2 COMMENT
      02 UPDATE LEVEL
0001 UPDATE PATCH TOTAL
0001 CURRENT PATCH TOTAL
```

The patch line has the following meaning: a patch identified by the patch-id G5195A was applied to overlay 01 by Honeywell update 02. The patch was applied to location 8C02E2; this location previously contained 0000, and now contains 0F02. The symbolic instruction and comment are also listed. Applicable summary lines follow the patch line.

### Example 2:

LP

Assuming an object unit is being patched, this example produces a printout of a listing of patches applied to the file. The printout would appear as:

```
NUMBRF 00000162 00000444 DC 444 COMMENT
      00000163 00000222 DC 222
NUMBRH 000001A6 00000333
      000001A7 00000444
      000001A8 <00000221
      000001AA 00000004
      000001AB 00000321
```

The printout has the following meaning: patch-id, address at which the patch was applied, and the patch value. Any symbolic instruction and comment entered on an SP directive are also listed. The first line designates that patch 0444, whose patch-id is NUMBRF, was applied to location 162. Note that the last two characters of the patch-id (e.g., RT) were omitted from the printout. The less than character (<) beside the patch value indicates the two-word relocatable address (IMA) 00000221 was patched in at locations 1A8 and 1A9.

## LIST PATCHES NOW

### LIST PATCHES NOW (LN)

List all patches on the specified file and then allow more patches to be applied. The listing is produced on the user-out file. This directive is effective only in the absentee (batch) mode and can be applied only to bound unit files. It must be the first directive issued. If it is entered in the interactive mode, entered for an object unit, or not the first directive entered, it is processed the same as an LP directive. The LN directive allows the current patches to be listed first and then additional patches to be applied without reloading Patch.

If patches on a bound unit that is distributed by Honeywell are being listed, the listing may also include update summary information. The information consists of the current update level of the bound unit and the total number of update- applied patches. A warning is issued if the total number of patches entered through updates is not equal to the actual count of patches on the bound unit.

#### FORMAT:

LN

#### Example:

LN

In this example, a printout of the patches applied to the bound unit is produced:

```
0000 CONRCT 038000A8 00005A4D 00005A4E DC 5A4E COMMENT
0001 CURRENT PATCH TOTAL
```

The patch line has the following meaning: a patch identified by the patch-id CONRCT was applied to overlay 00. The patch was applied to location 38000A8; this location previously contained 5A4D, and now contains 5A4E. Any symbolic instruction and comment entered on the SP directive are also listed. The printout of patches ends with a line providing the number of patches currently on the bound unit.

# LIST PATCH NAMES

## LIST PATCH NAMES (LS)

List the names (patch-ids and group-ids) of the patches in the specified file and exit from Patch if in absentee (batch) mode. The listing is produced on the user-out file. Addresses, values, and symbolics are not listed.

If patches on a bound unit that is distributed by Honeywell are being listed, the listing may also include update summary information. The information consists of the current update level of the bound unit and the total number of update- applied patches. A warning is issued if the total number of patches entered through updates is not equal to the actual count of patches on the bound unit.

### FORMAT:

LS

Example 1:

LS

Assuming patches to a bound unit distributed and updated by Honeywell is being listed, this example produces a printout of the names of all patches and patch groups applied to the bound unit. The printout would appear as:

```
0000 PATCH0 01
RT  PATCH1 02
```

```
02 UPDATE LEVEL
0002 UPDATE PATCH TOTAL
0002 CURRENT PATCH TOTAL
```

The printout has the following meaning: the patch identified by patch-id PATCH0 was applied to overlay 00 by Honeywell update 01 and the patch identified by patch-id PATCH1 was applied to the root by Honeywell update 02. Applicable summary lines follow the patch information.



LIST PATCH NAMES

Example 2:

LS

Assuming an object unit is being patched, this example produces a printout of the names of the patches applied to the object unit. The printout would appear as:

PATCH1  
PATCH2

The printout has the following meaning: the patches identified by patch-id PATCH1 and PATCH2 were applied to object unit file.

# LIST SPECIFIED PATCH

## LIST SPECIFIED PATCH (LS)

List patches for those patch-ids specified and exit from Patch if in absentee (batch) mode. The listing is produced on the user-out file. Up to five patch-ids can be requested per absentee run.

### FORMAT:

```
LS patch-id[,patch-id...]
```

### ARGUMENTS:

patch-id

Patch-id of the patch(es) to be removed. A patch-id comprises eight to ten characters: the first six can be any ASCII characters except spaces. The last two to four characters must identify the root or overlay to which the patch(es) are applied. If an object unit or the root or separate data area of a bound unit is being patched, the patch-id is eight characters, the last two of which are RT. If an overlay is being patched, the last two to four characters identify the hexadecimal overlay number. The first overlay is 00 for bound units created by the Linker (01 if linked with the -R option), and subsequent overlays are numbered consecutively in ascending order. There can be no embedded blanks. Within the root and each overlay, patch-ids must be unique.

### Example 1:

```
LS NUMBRART,NUMBRB00
```

Assuming a bound unit is being patched, this example produces a printout of the entire patch NUMBRART in the root and the entire patch NUMBRB00 in overlay 00.

### Example 2:

```
LS PATCH1RT,PATCH2RT
```

Assuming an object unit is being patched, this example produces a printout of the entire patch PATCH1 and the entire patch PATCH2 in the object unit.

# LIST UPDATES

## LIST UPDATES (LU)

List the update information of a bound unit that was distributed by Honeywell. The listing is produced on the user-out file. The information will consist of the update numbers of all updates applied to the bound unit and how many patches comprise each update. Update summary information consists of the current update level and the total number of update-applied patches. A warning is issued if the total number of patches entered through updates is not equal to the actual count of patches on the bound unit.

FORMAT:

LU

Example:

LU

Assuming patches to a bound unit distributed and updated by Honeywell is being listed, this example produces a printout of the update information. The printout would appear as:

```
01 UPDATE 05 PATCHES
02 UPDATE 03 PATCHES
02 UPDATE LEVEL
0008 UPDATE PATCH TOTAL
0008 CURRENT PATCH TOTAL
```

If the total number of patches entered through updates is not equal to the actual count of patches on the bound unit, the printout would appear as:

```
01 UPDATE 05 PATCHES
02 UPDATE 02 PATCHES
02 UPDATE LEVEL
0007 UPDATE PATCH TOTAL
0008 CURRENT PATCH TOTAL    *** WARNING ***
```

If no updates were applied to the bound unit, the printout would appear as:

```
NO UPDATES
0008 CURRENT PATCH TOTAL
```

# QUIT

## QUIT (Q)

Inform Patch that the final Patch directive has been entered, and initiate processing of the previous Patch directives if mode is absentee (batch). This directive is usually preceded by at least one other directive. When the directive(s) have been processed, execution of Patch terminates.

### FORMAT:

{ Q }  
{ QT }

## SET GLOBAL SHARE BIT OFF

### SET GLOBAL SHARE BIT OFF (GNSH)

Indicate that the patched bound unit is not globally shareable, which means that the program is not sharable between groups. This directive turns off the global share bit in the bound unit header area. The share bit is not affected by this directive. This directive cannot be used for object unit files.

The global share bit is set at link time by the GSHARE Linker directive.

#### FORMAT:

GNSH

#### Example:

GNSH

In this example, the bound unit is not globally sharable.

## SET GLOBAL SHARE BIT ON

### SET GLOBAL SHARE BIT ON (GSHR)

Indicate that the patched bound unit is globally sharable, which means that the program is sharable by all users on the system, and the root is loaded into the system memory pool. This directive turns on the global share and share bits in the bound unit header area and sets write access in the load unit descriptor (LUD) for the root and all fixed overlay segments to ring 0. The bound unit must have reentrant code.. Patch alters the status of the global share and share bits only; it makes no check on the sharability of the bound unit. This directive cannot be used for object unit files.

This Patch directive is equivalent to the Linker GSHARE directive.

#### FORMAT:

GSHR

#### Example:

GSHR

In this example, the bound unit is globally sharable and its root will be loaded into the system memory pool.

## SET SHARE BIT OFF

### SET SHARE BIT OFF (NS)

Indicate that the patched bound unit is not sharable within a memory pool. This directive turns off the global share and share bits in the bound unit header area and sets write access in the load unit descriptor (LUD) for the root and all fixed overlay segments to ring 3. This directive cannot be used for object unit files.

The share and global share bits are set at link time by the SHARE and GSHARE Linker directives.

#### FORMAT:

NS

#### Example:

NS

In this example, the bound unit is not sharable.

# SET SHARE BIT ON

## SET SHARE BIT ON (SS)

Indicate that the patched bound unit is sharable within a memory pool. This directive turns on the share bit in the bound unit header area and sets write access in the load unit descriptor (LUD) for the root and all fixed overlay segments to ring 0. The bound unit must have reentrant code. Patch alters the status of the share bit only; it makes no check on the sharability of the module. This directive cannot be used for object unit files.

This Patch directive is equivalent to the Linker SHARE directive.

### FORMAT:

SS

### Example:

SS

In this example, the bound unit is sharable. If another task requests that the bound unit be loaded, instead of another copy of the bound unit being loaded, the existing copy in memory is used.



SET SYSTEM BIT ON (STSY)

Indicate that the patched bound unit may run as a system task while clearing any indicators requiring the bound unit run in the user pool, the page pool, or the swap or page pool. This directive alters indicators in the bound unit header area. The directive cannot be used for object unit files.

Before using this directive, consult with the person responsible for system building and determine the available system memory. This Patch directive is equivalent to the Linker SYS directive.

FORMAT:

STSY

Example:

STSY

In this example, the bound unit header contains an indicator signifying that the bound unit may run as a system task.

# SYMBOLIC DATA PATCH

## SYMBOLIC DATA PATCH (SD)

Convert one or more symbolic instructions into the form of a hexadecimal patch and apply to an object unit or bound unit. For bound units, the directive causes patches to be applied to the separate data section of a -R linked bound unit. For object units, the directive causes one or more one-word instructions to be applied to common areas; i.e., to either named or local common blocks. You can verify the current contents of locations while patching.

### FORMAT:

For Bound Units -- No Verification:

```
SD patch-id;/off1;patchval1[;patchval2...][;/off2;patchval3...]
```

For Bound Units -- With Verification:

```
SD patch-id;/off1;(oldval1;newval1[oldval2;newval2...])  
[;/off2;(oldval3;newval3[;oldval4;newval4...])...]
```

For Object Units -- Named Common Block -- No Verification:

```
SD patch-id;blockname;/offs;patchval1[;patchval2...patchvaln]  
[;/offs;patchval1[;patchval2...patchvaln]...]
```

For Object Units -- Named Common Block -- With Verification:

```
SD patch-id;blockname;/offs;(oldval1;newval1  
[oldval2;newval2)...][;/offs;(oldval3;newval3  
[oldval4;newval4...])...]
```

For Object Units -- Local Common Block -- No Verification:

```
SD patch-id;/offl;patchval[;/offl;patchval]...
```

For Object Units -- Local Common Block -- With Verification:

```
SD patch-id;/offl;(oldval;newval)[;/offl;(oldval;newval)]...
```

## ARGUMENTS:

## patch-id

Patch-id of the patch(es) to be applied. A patch-id comprises eight characters; the first six can be any ASCII characters except spaces. The last two characters must be RT for an object unit or the separate data area of a bound unit linked with the -R option.

## offn

Relative location in the bound unit segment at which the first (or only) subsequent patch value is applied. Each address must comprise one to eight right-justified hexadecimal characters and must be preceded by the slash character (/). Subsequent patch values, if any, are applied to succeeding memory locations. A patch can have a maximum of 127 /addr fields and a maximum of 127 values for any /addr field. The segment base is added to all locations specified or implied and all relocatable addresses (IMAs).

## NOTE

Take care in specifying an address to be patched. If the address of a location to be patched is identified when a bound unit is being executed, that memory address has three possible factors:

- The original address of the location in the bound unit relative to the beginning of the bound unit
- The linking relocation factor
- The loader relocation factor.

If a bound unit address that is to be identified at execution time is being patched, the loader relocation must be subtracted from the address. If an object unit is being patched, both the linking and loader relocation must be subtracted. Object unit locations can also be obtained from the listing produced during assembly.

## offl

Non-negative offset from beginning of the local common block.

## SYMBOLIC DATA PATCH

### oldval

Verification value; one to four hexadecimal characters specifying the value that should be in the location before the patch is applied. If patchval is a relocatable address, verval can be one to eight hexadecimal characters.

### patchval (object units)

Value to insert into the common block. Relocatable address are not permitted. For a local common block, only one patch value can be specified for each offset.

The value must be specified as:

opcode field<sub>1</sub>[,field<sub>2</sub>][,field<sub>3</sub>]

opcode specifies a symbolic instruction; field<sub>n</sub> specifies either a register or a hexadecimal value.

### blockname

Symbolic name of the common block. The name can contain one through six characters.

### offs

Offset from the symbolic name of the common block.

### patchval (bound units)

Value to be inserted at an address, replacing the contents of that location. The value must be specified as a symbolic instruction.

### newval

Specify the patch value to be applied. See the appropriate description of patchval, above.

# SYMBOLIC PATCH

## SYMBOLIC PATCH (SP)

Convert one or more symbolic instructions into the form of a hexadecimal patch and apply to an object unit or bound unit. You can verify the current contents of locations while patching.

### FORMAT:

Without Verification:

```
SP patch-id[;base1]/addr1;instruction[;instruction...]  
[[+base2]/addr2;instruction[;instruction...]][%comment]
```

With Verification:

```
SP patch-id[;base1]/addr1;(oldval;instruction[;oldval  
instruction...])[;[+base2]/addr2;oldval;instruction  
[oldval;instruction...]][%comment]
```

### NOTES

1. One or more lines of arguments can be specified. When two or more lines of arguments are entered in an SP directive, instructions and verification values must not be split between lines. No line can begin or end with a semicolon (;). Individual fields, values, and addresses must not be split between lines. The entry of a Patch directive name (e.g., EP, LP) at the beginning of a line designates the end of the previous Patch directive. Hexadecimal patches are not permitted.
2. You can use a carriage return instead of a semicolon as a delimiter. Commas and spaces are not permitted because they are legitimate in symbolic instructions.
3. You can mix verification and nonverification patches. For example:

```
SP NUMBRDRT;/135;(111;LDV $R1,1;2;CL =$R2)  
/150;STB $B2,400
```

Only the patches at locations 135 and 136 are verified.

## SYMBOLIC PATCH

### ARGUMENTS:

#### patch-id

Patch-id of the patch(es) to be applied. A patch-id comprises eight to ten characters; the first six can be any ASCII characters except spaces. The last two to four characters must identify the root or overlay to which the patch(es) are being applied. If an object unit or the root of a bound unit is being patched, the patch-id is eight characters, the last two of which must be RT. If an overlay is being patched, the last two to four characters identify the hexadecimal overlay number. The first overlay is 00 for bound units created by the Linker (01 if linked with the -R option), and subsequent overlays are numbered consecutively in ascending order. There can be no embedded blanks. Within the root and each overlay, patch-ids must be unique.

#### base

Optional argument allowed only for bound units. Base defines a value that is added to all locations, i.e., /addr specified or implied and all relocatable addresses (IMAs). If this argument is omitted, the default base is the segment base. Base can be entered as a hexadecimal address of one to eight characters or as a name that has been specified as an EDEF at link time and placed in the bound unit symbol table. If a symbol name is used, Patch finds the name in the symbol table and uses its address as the base value. The format for the symbol name as a base is +symname, where symname comprises 1 to 12 characters. Except in the case of multiple base fields, if a hexadecimal address is used for base, the plus sign is not required. Leading hexadecimal digits A through F should be preceded by 0 to assure treatment as a numeric base.

For bound units created by the Linker the values specified for the /addr fields and IMA references (if any) must include the displacement of the root or overlay. This displacement is equal to the base address of the root or overlay as printed on the link map. The user can specify the base argument in the Patch directive, omit it and rely on the default base value, or specify a zero base and add the displacement to each /addr field and IMA to achieve the same result. For example, if the first overlay of a bound unit is based at 1000 and a patch to locations 100 to 103 and 200 to 204 is to be made within the overlay, the following patch directives are equivalent:

```
SP NUMBRA00;0;/1100;LDR $R1,1500;STR $R1,=$R2
/1200;ADD $R1,1600;JMP 1156
```

```
SP NUMBRA00;1000;/100;LDR $R1,500;STR $R1,=$R2
/200;ADD $R1,600;JMP 156
```

```
SP NUMBRA00;/100;LDR $R1,500;STR $R1,=$R2
/200;ADD $R1,600;JMP 156
```

There can be multiple base values in one directive line. The first of the multiple base values can optionally be preceded by a plus (+) sign; the remaining base values must be preceded by a plus sign, as shown:

```
SP NUMBRART;100;/10;LAB $B1,100
+0;/40;STR $R1,<100
+10;/60;ADV $R4,3
```

/addr

Relative location at which the first (or only) subsequent patch value is applied. Each address must comprise one to eight right-justified hexadecimal characters and must be preceded by the slash character (/). Subsequent patch values, if any, are applied to succeeding memory locations. A patch can have a maximum of 127 /addr fields and a maximum of 127 values for any /addr field.

## SYMBOLIC PATCH

### NOTE

Take care in specifying an address to be patched. If the address of a location to be patched is identified when a bound unit is being executed, that memory address has three possible factors:

- The original address of the location in the bound unit relative to the beginning of the bound unit
- The linking relocation factor
- The loader relocation factor.

If a bound unit address that is to be identified at execution time is being patched, the loader relocation must be subtracted from the address. If an object unit is being patched, both the linking and loader relocation must be subtracted. Object unit locations can also be obtained from the listing produced during assembly.

### instruction

Value to be inserted at an address, replacing the contents of that location. The value must be specified as:

opcode field<sub>1</sub>[,field<sub>2</sub>][,field<sub>3</sub>]

opcode specifies a symbolic instruction (except for I/O or floating-point instructions); field specifies either a register or a hexadecimal value.

Field can contain a 1 to 12 character symbol name that has been specified as an EDEF at link time. Name must be preceded by an exclamation mark (!).

Field can contain positive or negative offsets with the dollar sign (\$) indicating the current address. All offsets must be hexadecimal values.

SP NUMBRART;/100;B \$+12;B \$-2F;B >\$+5



## SYMBOLIC PATCH

Value may be a text statement of up to 30 characters. Field must be enclosed in single quotes ('). If verify values are used, the number of verify values must equal the length of the text in words. Text must be an even number of characters (i.e., if the text is an odd number of characters, then an extra space character should be used).

```
SP NUMBRART;/100;TEXT 'AMESSAGE ONE'
```

Field can use the symbol ZHCOMM preceded by an exclamation mark (!) to represent unrelocated zero. In this case the less than character (<) indicates two words and not relocatable; without it the address is a displacement.

```
SP NUMBRART;/100;LAB $B1,<!ZHCOMM+5  
LDB $B4,<!ZHCOMM;STR $R1,!ZHCOMM+5
```

oldval

Verification value; one to four hexadecimal characters specifying the value that should be in the location before the patch is applied. If patchval is a relocatable address, verval can be one to eight hexadecimal characters.

comment

After the last character in a directive line, a space followed by a percent sign (%) or a 0,8,4 punch on a card causes Patch to interpret the rest of the line as a comment. The percent sign is replaced by a blank. Parentheses are not allowed in comments on patches specifying verification. A comment must not be split between lines; i.e., it must end the line. Comments are written along with patches to patch history records. Therefore, when patches are listed (via the LP or LN directives), comments are listed also.

Example:

```
SP NUMBRART;/100;LDV $R1,1 %THIS IS A COMMENT
```

Comments significantly increase the amount of media space taken up by patch history records at the end of object and bound units.

# VDEF

## VDEF

Assign a specified value to an undefined external symbol within a bound unit and change all locations that reference this symbol to the specified value. This directive cannot be used for object unit files.

### FORMAT:

```
VDEF symname,value[,L]
```

### ARGUMENTS:

symname

Name of the external reference that is assigned a value; can be from 1 to 12 characters in length.

value

Value that is assigned to all references to symname.

L

List all changed references to symname on the device specified as user-out. Default: No list.

### Example:

```
VDEF VALZZ,50,L
```

This example assigns the value 50 to the undefined external symbol VALZZ, changes all locations that referenced VALZZ to 50, and lists all changed locations.

VDEF is used for changing undefined value definitions. The LDEF directive is used for changing undefined location definitions.

Undefined external references in a bound unit can be defined by a VDEF directive only one time. If you make a mistake you must use HP, DP, SP or SD directives to change each location containing the incorrectly defined value. No listing of the VDEF Patch processing is kept, therefore, the L argument should be used.

REMOVE THIS PAGE AND PLACE TAB FOR

TAB 11

MESSAGES



## *Section 11*

# **MESSAGES**

This section describes the Message Reporter and how to add user messages to the message library.

### MESSAGE REPORTER

The Message Reporter is a system service that is used to retrieve and display prepared messages stored in a library file on disk. It is used by system components to display messages that indicate error conditions or give help to the user of the terminal, and it can be used for these purposes by application programs as well. You can invoke the Message Reporter either through program calls or through the Display command.

If the program running at the terminal is being executed under control of the command processor, the Message Reporter sends messages to the terminal via the error-out file. If the program operates the terminal in text mode, the messages are displayed starting at the current cursor position. If the program operates the terminal in forms mode, the messages are displayed in the supervisory message line (line 24 on VIP7200 and VIP7700 terminals; line 25 on VIP7300 and VIP7800 terminals).

If the program is being executed under the control of the menu processor, the program's messages, as well as any help messages connected with the menus, are displayed in the message region, which is maintained by the menu processor. This comprises lines 20 through 23 on VIP7200 and VIP7700 terminals and lines 21 through 24 on VIP7300 and VIP7800 terminals. However, if the selection menu specifies that the program is to have control of the entire screen, its messages are displayed in the supervisory message line.

### Message Libraries

A message library is an indexed sequential disk file in which prepared messages are stored in ascending order according to a five-character record key. This key is known to programs using the library as the message code. When a program asks for a particular message, it passes the message code to the Message Reporter, which retrieves from the file the record having that key value. If the Message Reporter fails to find a record with that key value, it reports the message code instead of the message text.

You may have more than one message library in use at the same time. If you have an application that runs only once a week, you can put its specific messages on a separate message library rather than adding them to the Honeywell-supplied system message library. This conserves disk space when the application is not running.

### SYSTEM MESSAGE LIBRARY

A system message library file is supplied with the MOD 400 system to provide the text for the messages displayed by system components. Its pathname is:

>>ML>MLFILE.EN

where >> specifies location directly under the system root directory (as opposed to a user root directory). This is the default pathname that the Message Reporter uses to reference the system message library.

Some of the messages in the system message library are standard messages that may be of use to application programs. You may also add messages to the system message library.

### GROUP LIBRARIES

Message library file names conventionally have a suffix consisting of a period (.) followed by two alphabetic characters. This is called the "language key" because it is used to indicate the national language in which the library is written. The language key for the default system library MLFILE.EN indicates that the messages are in English.

An alternate language key can be specified in the Current Language Key entry in the user's profile file when the user is registered. That key then replaces the EN key in the pathname for the system message library file. Thus, for example, a message library written in French would be named MLFILE.FR, and users wishing to receive messages in French would have FR specified as their current language key. Instead of MLFILE.EN, the Message Reporter would reference MLFILE.FR.

The registered user can also enter or change the current language key entry in his or her profile file when logging in by including the -LK argument in the LOGIN command. The current language key (regardless of how it is entered in the profile file) becomes the default language key until changed.

The system commands that activate task groups (Enter Batch Request, Enter Group Request, and Spawn Group) support an argument (-ML pathname) that allows you to specify a message library to be used for all tasks executing within the group. If the Message Reporter fails to find a specified message record in the group library, it then searches the system library. If no -ML argument is entered when a task group is activated, its group library is the same as that for its parent process.

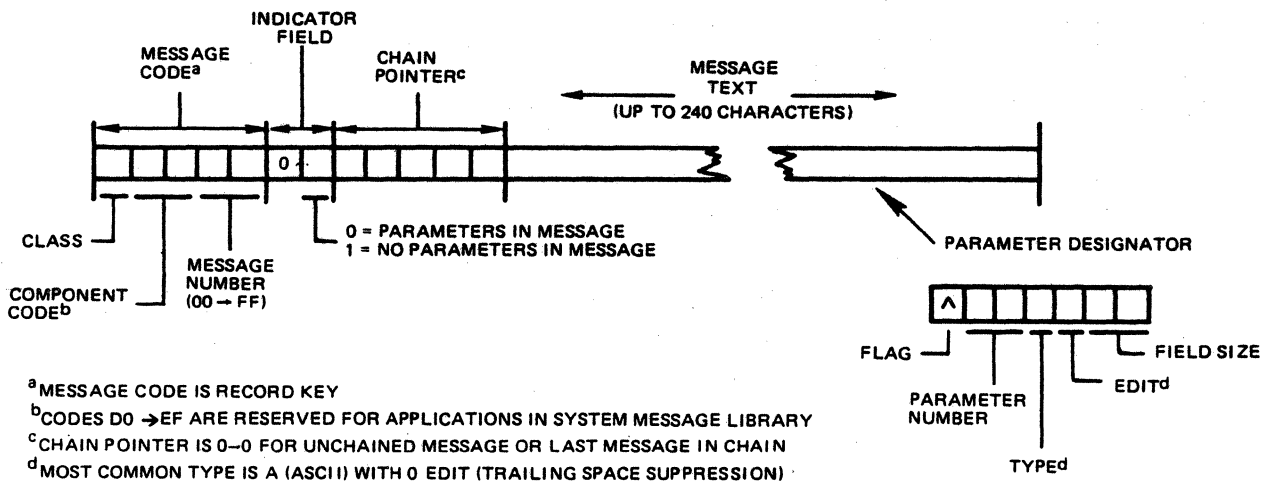
#### PRIMARY LIBRARIES

Tasks created within a group use the message library of the group unless it is changed by a Change Message Library (CML) command. After a task group is activated, the CML command can be used to enter the pathname for a "primary" message library to be searched before any other library associated with the task group. If the Message Reporter fails to find a message record in the primary library, it then searches the group library, if one has been specified. If it fails to find the record in either the primary library or the group library, it searches the system library. The CML command entered without a pathname removes the primary library assignment.

You can determine the primary message library being used by your program by using the List Message Library (LML) command.

#### Message Format

Each message in a message library file is stored as an ASCII character string in a variable-length record whose maximum length is 252 bytes. Figure 11-1 shows how message records are formatted; a brief explanation follows.



84-840-1

Figure 11-1. Message Library Record Structure

### MESSAGE CODE

The first five characters of the record are the key value, which is also the message code. This is an ASCII representation of a hexadecimal value. Conventionally, the first character is 0 for error messages and standard system messages, and 1 or 2 for help messages and other kinds of advice messages. Messages beginning with 3, 4, 5, 6, or 7 are parts of chains (see "Message Chaining" later in this section). The next two characters identify the component (or program) calling for the display of the message. Note that component codes D0 through EF in the system message library are reserved for application programs. The last two characters are used to select among the messages associated with a particular program. The range for these last two characters is 00 through FF.

### INDICATOR FIELD

The next two characters (bytes 6 and 7) are called the indicator field. If the text of the message includes parameters (see "Parameter Designators" later in this section), the value in the indicator field must be 00. This tells the Message Reporter to process the parameters. If the text of the message does not include parameters, the value here should be 01 (to prevent unnecessary execution of the code that processes parameters).

### CHAIN POINTER

The next five characters (bytes 8 through 12) can be used as the pointer to a subsequent message in a message chain. This pointer consists of the message code of the next piece of the message chain or it is 0000 (signifying the end of the chain).



## MESSAGE TEXT

The remaining characters are the text of the message. Up to 240 characters can be accommodated. This allows for up to three lines of 80 characters each. The Message Reporter displays the amount of text that will fit on a line before displaying text on the next line, unless the text contains a special 2 character string called a new line indicator. If the Message Reporter encounters a new line indicator, which is a circumflex (^) followed by a slash (/), the remainder of the text is displayed at the beginning of the next line.

### Parameter Designators

Messages can be formulated to include special character strings called parameter designators. A parameter designator in a message defines a substitution field: the Message Reporter replaces it with a character string supplied by the program calling for the display of the message. Thus, parameterization allows the formulation of generalized messages that can be tailored to particular situations by different programs.

### PARAMETERIZED MESSAGES

A process wanting to display a parameterized message calls the Message Reporter, passing it the message code as usual, but also passing it parameter values. The Message Reporter obtains the message from the library, and then performs the substitution of the parameter values for the parameter designators.

The passage of parameters to the Message Reporter is supported by macrocalls in Assembly language and by calling procedures in higher-level languages such as COBOL and FORTRAN.

Note that a parameterized message record must have 00 (ASCII) in its indicator field (bytes 6 and 7) in order to be processed as such. (If the indicator field is 01, the parameter designators are treated as normal text.) If a program calls for the display of a correctly parameterized message, but does not pass any parameter values, the Message Reporter removes the parameter designator(s) and closes up the remaining message text.

Note also that if message chaining is used (see "Message Chaining" later in this section), only the first message in a chain can be parameterized.

Here are two examples of parameterized message text. In these examples the message code, indicator, and chain pointer fields are omitted; the parameter designators are underscored, and upper case characters are used for clarity:

```
DELETING TASK GROUP ^01A029 AT ^02A118.  
EXECUTION [OF ^01A014] COMPLETED. PRESS CLEAR TO CONTINUE.
```



A special record in the system message library allows you to substitute other characters for the circumflex and the brackets. The format for this message is:

```
000FF0100000^a[b]c
```

where a is the substitute character for the circumflex, b is the substitute for the left bracket, and c is the substitute for the right bracket. The default content of this record is:

```
000FF0100000^^[[[]]
```

If this record is altered, all messages on the library that use these characters must also be changed.

#### Parameter Number

The two numeric characters following the flag are the parameter's number--in effect, its address within the message. Every parameter designator includes a parameter number in the range of 01 to 99, although its usefulness is evident only when the message provides for two or more parameters. Assembly language programs pass message parameters by means of a data structure in which the parameters are identified by number, and therefore such programs have random access to the parameter designators in a message. Given a message like this:

```
ERROR IN [^01A015 RECORD OF] ^02A015 FILE.
```

an Assembly language program could omit a value for parameter #01, and specify a value for parameter #02, in which case the message might be displayed as:

```
ERROR IN SMITH FILE.
```

A similar message:

```
THE ^02A015 FILE STILL HAS AN ERROR! [REENTER RECORD ^01A015.]  
could be displayed as:
```

```
THE SMITH FILE STILL HAS AN ERROR!
```

or as:

```
THE FILE STILL HAS AN ERROR!
```

or as:

```
THE SMITH FILE STILL HAS AN ERROR! REENTER RECORD 00234.
```

or as:

```
THE FILE STILL HAS AN ERROR! REENTER RECORD 00234.
```

depending upon the parameter(s) for which the calling program supplied values. Note the implication that parameters in a message do not have to be ordered by parameter number.

For programs written in a higher-level language, the use of messages having more than one parameter is somewhat more restricted because the statements that support calls to the Message Reporter require the values for the parameters to be passed as positional arguments (refer to "Message Library Utilities" later in this section for more details). Thus, when programs written in these languages want to display messages having two or more parameters, the relative position of the argument in the statement determines the parameter designator to which the argument applies. This also means that parameters cannot be skipped. Here again are the two messages used above. For this illustration, they have message codes 12345 and 12346, respectively:

```
123450000000ERROR IN [^01A015 RECORD OF] ^02A015 FILE.  
123460000000THE ^02A015 FILE STILL HAS AN ERROR! [REENTER RECORD ^01A015.]
```

Given these two messages and the following COBOL data declarations:

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 MSG1 PIC XXXXX VALUE "12345".  
01 MSG2 PIC XXXXX VALUE "12346".  
01 PAR1 PIC 9(6) VALUE "000234".  
01 PAR2 PIC X(15) VALUE "SMITH".
```

these procedure statements:

```
CALL "ZXDSMG" USING MSG1 PAR1 PAR2.  
CALL "ZXDSMG" USING MSG1 PAR1.  
CALL "ZXDSMG" USING MSG1 PAR2.  
CALL "ZXDSMG" USING MSG2 PAR1 PAR2.
```

would display, respectively, these messages:

```
ERROR IN 000234 RECORD OF SMITH FILE.  
ERROR IN 000234 RECORD OF FILE.  
ERROR IN SMITH RECORD OF FILE.  
THE SMITH FILE STILL HAS AN ERROR! REENTER RECORD 000234.
```

#### Parameter Type

The character following the parameter number is alphabetic, and it specifies the parameter type, that is, the kind of information to be inserted in the message. In most cases, this character is the letter A, specifying that the parameter type is an ASCII character string.

The possible parameter types and their meanings are:

| <u>Parameter Type</u> | <u>Meaning</u>                    |
|-----------------------|-----------------------------------|
| A                     | ASCII Character String            |
| B                     | Bit String                        |
| D                     | Convert Hexadecimal to Decimal    |
| H                     | Hexadecimal                       |
| P                     | Address Pointer                   |
| R                     | Radix 40 Packed Name              |
| T                     | Convert Internal Time to External |

### Edit

The character following the parameter type designator is numeric and it specifies the editing to be performed on the parameter value. The implication of the edit character depends upon the parameter type specification. However, assuming that the parameter type is A (an ASCII character string), the value here is most often 0, which specifies trailing blank suppression. Thus, if the ASCII character string you want to pass to the Message Reporter does not fill the buffer you have allocated, the trailing blanks are suppressed (do not occupy space in the message).

The possible edit character values for each of the parameter types are:

| <u>Parameter Type</u> | <u>Edit Character and Meaning</u>  |
|-----------------------|--|
| A or R                | 0= Suppress Trailing Blanks<br>1= Don't Suppress Trailing Blanks   |
| B                     | Bit Offset to the Beginning of String  |
| D                     | 0= Signed, Suppress Leading Zeros, Left-Justified<br>1= Unsigned, Suppress Leading Zeros, Left-Justified<br>2= Signed, No Zero Suppression<br>3= Unsigned, No Zero Suppression<br>4= Signed, Suppress Leading Zeros, Right-Justified<br>5= Unsigned, Suppress Leading Zeros, Right-Justified |

### NOTES

1. The sign precedes the digits only if the sign is negative. Therefore, negative numbers produce one extra character, which should be included in the field size.
2. No zero suppression implies right-justification with left zero fill. If the number is signed and negative, the sign replaces the leftmost zero.

Parameter  
Type                      Edit Character and Meaning

- H            0= Don't Suppress Leading Zeros  
             1= Suppress Leading Zeros
- P            Must Be Zero
- T            0= Display Date/Time (yyyy/mm/dd hhmm:ss.mmm)  
             1= Display Time Only (hhmm:ss.mmm)  
             2= Display Elapsed Time (hhhh:mm:ss.mmm)

Field Size

The last two numeric characters specify the number of character positions that the parameter field allows. For ASCII character strings, the maximum field size is 80 characters. The maximum field size for each of the parameter types is:

| <u>Parameter</u><br><u>Type</u> | <u>Units</u>       | <u>Range</u> | <u>From Caller</u> |
|---------------------------------|--------------------|--------------|--------------------|
| A                               | Characters         | 1 to 80      | 1 to 80 bytes      |
| B                               | Bits               | 1 to 32      | 1 to 32 bits       |
| D                               | Decimal Digits     | 1 to 10      | 1 to 4 bytes       |
| H                               | Hexadecimal Digits | 1 to 8       | 1 to 4 bytes       |
| P                               | Hexadecimal Digits | 6            | 4 bytes            |
| R                               | Characters         | 1 to 80      | 1 to 80 bytes      |
| T                               | Characters         | 1 to 22      | 6 bytes            |

Message Chaining

Message chaining is the method by which the "more help?" function is implemented. If "more help?" is enabled through the MHON (More Help On) command, and the Message Reporter displays a message whose chain pointer is nonzero, it also displays the "more help?" prompt. The display of the first message may have been initiated by a program (e.g., to report an error), or by the user having pressed the HELP key while a form or menu was displayed. In either case, if the user responds affirmatively to the "more help?" prompt, the Message Reporter displays another related message. This continues until either the user responds negatively to the "more help?" prompt, or the end of the sequence of messages is reached, whereupon the Message Reporter so indicates with an end-of-message message.

This function is enabled by the MHON (More Help On) command, and disabled by the MHOFF (More Help Off) command. It is normally enabled by default for most processes at system startup. It is not automatically enabled for the execution of EC commands.

Given the following library, and assuming that "more help" is enabled, the chaining function would work as follows:

```
000AA0100000yes
000CF0100000Do you wish more help? (Type yes or no.)
000DF0100000Press HELP key for more help; press XMIT key to exit help.
000EF0100000End of message
188000158801This is the first message in the chain
588010158802This is the second message in the chain
588020100000This is the last message in the chain.
```

The chain pointer (bytes 8 through 12) of message 18800 contains the message code 58801. Message 58801 points to message 58802 in the same manner. Message 58802's pointer is zero, which specifies that it is the end of the chain.

When the Message Reporter receives a request to display message 18800, it observes that the 18800's pointer is nonzero. After displaying message 18800, it displays the "more help?" prompt. If the calling program is operating the terminal in TTY mode, the "more help?" prompt is message number 000CF, and the affirmative response is typing Yes (actually, just the letter Y). The Message Reporter compares the user's response with the first character in message 000AA, which in this case is the letter Y. (For a national language other than English, some other character could be substituted in message record 000AA.) Any other key-stroke is interpreted as a negative response.

If the terminal is being operated in forms mode, the "more help?" prompt is message 000DF, and the affirmative response is the user pressing the HELP key; the negative response is the user pressing the CLEAR key.

In either case, if the user responds affirmatively, the Message Reporter then displays message 58801, and repeats the "more help?" prompt. If the user again responds affirmatively, the Message Reporter displays messages 58802 and 000EF. Any user response terminates the function at this point.

Note that message 000AA must be present in the library if message chaining is to work; if it is not, the Message Reporter does not display the "more help?" prompt.

Only the first message in a message chain can be parameterized. Subsequent messages should have their indicator fields set to 01 (ASCII) to avoid unproductive execution of the parameter processing code.

## Standard Messages in the System Message Library

The following standard messages are supplied in the system message library; they have a "00" component code and are available for any program to use:

0002B   \*\*\*STANDARD MESSAGES\*\*\*\*

000AA   yes

000AB   no

000AC   No interactive help available.

000AD   Argument ^01A065 not recognized.

000AE   Parameter ^01A065 not recognized.

000AF   Please answer "yes" or "no".

000B0   Key is inactive at this time.

000B1   Monday^Tuesday^Wednesday^Thursday^Friday^Saturday^  
Sunday^

000B2   January^February^March^April^May^June^July^August^  
September^October^November^December^

000B3   Jan^Feb^Mar^Apr^May^Jun^Jul^Aug^Sep^Oct^Nov^Dec^

000B4   [^01A006] Called with an incorrect number of  
parameters

000B5   True

000B6   False

000CE   Press XMIT to continue [-message ^01A005-]

000CF   Do you wish more help? (Type yes or no.)

000DF   Press HELP key for more help; press XMIT key to exit  
help.

000FF   ^^[[[]]

If messages B1 through B3 are used, the user must determine the correct separator character via message FF. Messages AD, AF, and B4 allow for a specialization parameter (65 ASCII characters for messages AD and AF; 6 ASCII characters for message B4).



## Message Library Utilities

Higher-level languages such as COBOL, FORTRAN, BASIC, and PASCAL can use the ZXDSMG and ZXBTMG utilities to send and receive messages. ZXDSMG allows a program to send a specified message to the error-out file. ZXBTMG allows a program to place a message in a specified buffer.

### ZXDSMG UTILITY

The ZXDSMG utility enables an application to send a specified message to the error-out file. The message must be in one of the message libraries, and can have parameters.

ZXDSMG accepts arguments of the following form:

- <msg\_num> - Set of five ASCII characters that specify the message code.
- <arg1> - Optional list of specialization parameters.
  - . Each is a set of ASCII characters.
  - .
  - .
- <argn>

### ZXBTMG UTILITY

The ZXBTMG utility returns a message to a specified buffer. The message can contain parameters. If the buffer is not large enough to contain the entire message, it is filled with asterisks.

ZXBTMG accepts arguments of the following form:

- <msg\_num> - Set of five ASCII characters that specify the message code.
- <buffer> - A block of memory set aside to receive the message specified by <msg\_num>.
- <arg1> - Optional list of specialization parameters.
  - . Each is a set of ASCII characters.
  - .
  - .
- <argn>

## ARGUMENT DEFINITION

You must define the previously described arguments according to the requirements of the language you are using. For example, to define <msg\_num>:

| <u>Language</u> | <u>Argument Definition</u>  |
|-----------------|---|
| COBOL           | MES-NUMB PIC X(5) VALUE "0021F"   |
| FORTRAN         | CHARACTER*5 MES_NUMB<br>.<br>.<br>.<br>MES_NUMB = '0021F'                     |
| BASIC           | MESNUMB\$ = '0021F'   |
| PASCAL          | MES_NUMB: PACKED ARRAY [1..5] OF CHAR;<br>.<br>.<br>.<br>MES_NUMB: = '0021F'; |

Refer to the appropriate language manual for detailed information on how to reference external programs.

## UPDATING THE MESSAGE LIBRARY

The message library allows you to create error, status, text, and help messages and to store the messages as described below.

### Message Structure

User-created messages to be stored in the message library must have the following structure (see Figure 11-1):

- A hexadecimal, five-character message code (stored as five ASCII bytes), in the form nyyzz. The value for n must be from 0 through F. In the system message library, the value for yy must be selected from the component message codes D0 through EF (hexadecimal), which are reserved for user definition. (Note, however, that TPS 6 and QR6 use component message codes 80 through CF. Users who run TPS 6 and QR6 must not use these component codes for user-created messages.) The value for zz can be anywhere in the range 00 through FF.
- Two-character indicator word

- Five-character chain pointer
- Message text. This text cannot exceed 240 ASCII characters. The message text should be left-justified. Messages can be displayed with the message identifier preceding the text. In this case, the initial piece of the message cannot exceed 216 ASCII characters.

The system message library file name is >>ML>MLFILE.xx, where xx is a language suffix.

### Adding A Message to the Message Library

Use the Line Editor or Screen Editor to create a file (an abbreviated version of the message library) to change messages in the master message library. The Add/Delete Message utility (ADM, described in the MOD 400 Commands manual) provides the ability to add, update, or delete messages on a message library using the abbreviated message library created with the Line Editor or the Screen Editor.

When updating the message library, the user program or the Display command may be used to check the message text. The Display command may also be used to generate a list of the available component codes. See the GCOS 6 MOD 400 Commands manual (CZ17) for a description of the Display command.

First an indexed sequential file is created. The file must be UFAS-indexed sequential, and have the following characteristics:

```

variable length
indexed file organization
logical record length      - 252
control interval size     - 512
allocation growth size    - 8
number of key descriptors - 1
number of key components  - 1
key type                  - C
key component size        - 5
key component offset      - 1

```

To create the message file, use the following command:

```
CR name -LRSZ 252 -GRSZ 8 -KSZ 5 -KLOC 1 -IND
```

Prior to using either of the editors to create the message records, the Set Terminal Characteristics (STTY) command is used to change the terminal line length to the maximum message record size (252 characters).

#### NOTE

The Set Terminal Characteristics command is not valid in the \$S group.

The editor is invoked once the terminal line length is changed. To invoke the Line Editor, use the following command:

ED -PT -LL 252 -NBS

The entire message library file is ASCII. The format of each of the message entries is as follows:

Message Code (5 characters)

Character 1 - ASCII 0 = error message  
(n) ASCII 1, 2 = help/text message  
ASCII 3-F = chained message text

Characters 2-5 - Four uppercase ASCII characters that represent a unique four-digit hexadecimal number in the range 0000 to FFFF.  
(yyzz)

Indicator Word - ASCII 00 - The message may have parameter substitution.

ASCII 01 - The message has no parameter substitution. This indicator must be set to 01 for chain messages.

Chain Pointer - Link to the next message in the chain; this is a five-digit ASCII number consisting of the message code of the next piece of the chain, or it is 00000 (signifying the end of the chain).

Message Text - ASCII message text optionally containing embedded parameter designators for the substitution of parameters.

To delete a message, including all parts of the chain, include only the message code of the first message in the chain.

The editors do not provide a means for automatically checking whether message chains have been added or deleted correctly. If the message is chained, all pieces of the chain must be submitted with each updated or added entry. Chained messages may have parameters substituted only in the first message of the series. The second and subsequent messages in the series must be entirely text (all parameter designators are ignored) and the message's indicator word must be set to 01.

This message file is supplied to the Add/Delete Message (ADM) program, which is used to update the message library. The Add/Delete Message utility cannot update the message library file currently in use; it must update a copy of the message library. A copy of the output from the ADM program indicates the status of each message (i.e., whether it was added, deleted, or rejected). The ADM program does not allow invalid messages to be added into the message library.

Even before the master message library is updated using the Add/Delete Message utility, new messages can be checked by your program or by means of the Display command by making the abbreviated message library the first library looked at by the task. This is accomplished by using the Change Message Library (CML) command. Once you are satisfied with the text of the messages you want to add to the message library you can use the Add/Delete Message (ADM) command to update your master message library.

NOTE

The Change Message Library command is not valid in the \$\$ group.

Examples:

| <u>Message Code</u> | <u>Indicator Word</u> | <u>Chain Pointer</u> | <u>Message Text</u>  |
|---------------------|-----------------------|----------------------|--|
| 00105               | 00                    | 00000                | DEVICE [!^01A005] NOT READY [CH = ^02H] [ST = ^03H]        |
| 00201               | 01                    | 00000                | The pathname violates naming conventions.                  |
| 0D000               | 00                    | 00000                | RECORD ^01H IS WITHHOLDING TAX                             |
| 0D001               | 00                    | 3D001                | USER-ID IS ^01A020   |
| 3D001               | 01                    | 3D002                | IT CONSISTS OF A NAME, PROJECT, AND MODE                   |
| 3D002               | 01                    | 00000                | NAME, PROJECT, AND MODE ARE UPPERCASE                      |
| 000CF               | 01                    | 00000                | Do you wish more help? (type yes or no)                    |
| 000DF               | 01                    | 00000                | Press HELP key for more help; press XMIT key to exit help. |
| 000EF               | 00                    | 00000                | [end of message ^01A005]                                   |
| 000FF               | 01                    | 00000                | ^^[[[]]  |

Optional parameters allow you to supply text for the message with your program. If the parameter is supplied, it is inserted in the proper location in the message. If a parameter is not supplied, the same message is displayed with the parameter designator and associated bracketed text eliminated from the displayed message. If a parameter is supplied and there is no parameter designator in the message, the parameter descriptor is ignored.

NOTE

If a message contains parameter designators but the indicator word is 01, the message is displayed with parameter designator characters even if a parameter is supplied.

## National Language Support

The Executive can operate with more than one message library; therefore, different users can execute with message libraries in different languages. Use the steps described earlier to build and update a message library in a different language.

### NOTE

Message 0392C may not be translated or made lower-case.

Message number 000FF is a special case on the system message library. On the English language library, the message is:

```
000FF0100000 ^^ [[]]
```

This message defines the special characters that are used to signal optional parameters to the system. If "^", "[", or "]" is not a valid special character you can either change message 000FF or change all supplied messages with parameter designators so that newly-defined special characters are used. For example, if "-" is to be used as a special character instead of the default "^" to indicate a parameter designator in a message, message 000FF becomes:

```
000FF0100000 ^- [[]]
```

Parameter designators can be in any order in a message. If a message is supplied with two designators, the message can be translated and the designators moved within the message with no effect on the programs supplying the missing parameters. For example,

```
text ^01A010 text ^02A012
```

can be translated into:

```
translated text ^02A012 translated text ^01A010
```

REMOVE THIS PAGE AND PLACE TAB FOR

TAB 12

MEMORY DUMPS





## MANUAL DIRECTORY

### MOD 400 OPERATING SYSTEM MANUALS

| <u>Base<br/>Publication<br/>Number</u> | <u>Manual Title</u>   |
|--|---|
| HE01                                   | ONE PLUS Guide to Software Documentation                            |
| CZ02                                   | GCOS 6 MOD 400 System Building and<br>Administration                |
| CZ03                                   | GCOS 6 MOD 400 System Concepts                                      |
| CZ04                                   | GCOS 6 MOD 400 System User's Guide                                  |
| CZ05                                   | GCOS 6 MOD 400 System Programmer's Guide -<br>Volume I              |
| CZ06                                   | GCOS 6 MOD 400 System Programmer's Guide -<br>Volume II             |
| CZ07                                   | GCOS 6 MOD 400 Programmer's Pocket Guide                            |
| CZ09                                   | GCOS 6 MOD 400 System Maintenance Facility<br>Administrator's Guide |
| CZ10                                   | GCOS 6 MOD 400 Menu System User's Guide                             |
| CZ11                                   | GCOS 6 MOD 400 Software Installation Guide                          |
| CZ15                                   | GCOS 6 MOD 400 Application Developer's Guide                        |
| CZ16                                   | GCOS 6 MOD 400 System Messages                                      |
| CZ17                                   | GCOS 6 MOD 400 Commands   |
| CZ18                                   | GCOS 6 Sort/Merge   |
| CZ19                                   | GCOS 6 Data File Organizations and Formats                          |
| CZ20                                   | GCOS 6 MOD 400 Transaction Control Language<br>Facility             |
| CZ21                                   | GCOS 6 MOD 400 Display Formatting and Control                       |
| CZ22                                   | GCOS 6 VISION Reference Manual                                      |
| GZ13                                   | GCOS 6 MOD 400 R3.1 to R4.0 Migration Guide                         |
| HC01                                   | GCOS 6 MOD 400 Application Development<br>Overview                  |
| HH42                                   | ONE PLUS System Administration                                      |
| HH58                                   | ONE PLUS System Services  |
| HH59                                   | ONE PLUS Menu Reference Card  |



## INDEX

- 9-Track
  - 9-Track Magnetic Tape File Organization, 3-11
- Abnormal
  - Clear Abnormal Trap Bit, 8-17
  - Turn On Abnormal Trap Bit, 8-68
- Absolute
  - Absolute and Relative Pathnames, 3-8
- Access
  - Procedures and Conventions After Access, 2-8
  - User Access Procedures, 2-1
- Active
  - Determining the Active Level, 8-12
  - No Level Active at the Time of Dump, 9-22
  - User Level Active at the Time of Dump, 9-22
- Adding
  - Adding a Message to the Message Library, 11-15
  - Adding and Deleting Lines, 5-120
  - Adding Lines to the Current Buffer, 5-123
- Address
  - Address Prefix, 5-93
  - Designating a Line Number as an Address, 5-6
  - Designating Contents of Line as an Address, 5-7
- Addresses
  - Compound Addresses, 5-11
  - Methods of Specifying Addresses, 5-5
- Appending
  - Appending a New String to an Existing String, 5-123
  - Appending Lines, 5-124
- Application
  - Application Development Components (Fig), 1-3
  - Application Development Components, 1-2
- Automatic
  - Automatic Tape Volume Recognition, 3-12
- Auxiliary
  - Auxiliary Buffer Directives and Escape Sequences, 5-65
  - Current and Auxiliary Buffers, 5-125
- Avoiding
  - Avoiding Post-Deletion Problems, 5-119
- BACKSPACE, 4-65
- Banner
  - Banner Login, 2-3
- Bit
  - Clear Abnormal Trap Bit, 8-17
  - Clear System Bit, 10-8
  - Set Global Share Bit Off, 10-35
  - Set Global Share Bit On, 10-36
  - Set Share Bit Off, 10-37
  - Set Share Bit On, 10-38
  - Set System Bit On, 10-39
  - Turn On Abnormal Trap Bit, 8-68
- Block
  - Block, 4-49
  - Block Description, 4-10
  - Change Block (Change Block or Cb), 4-21
  - Copy Block, 4-51
  - Delete Block, 4-53
  - Erase Block, 4-54
  - Move Block, 4-56
  - Write Block (Write Block or Wb), 4-44

## INDEX

### Bound Unit

- Clear All Bound Unit Breakpoints, 8-18
- Clear Bound Unit Breakpoint, 8-21
- Debugging Multiple Bound Units, 7-30
- Interrogate Bound Unit, 10-22
- List All Bound Unit Breakpoints, 8-39
- List Bound Unit Breakpoint Directive, 8-42
- Set Bound Unit Breakpoint, 8-55, 8-56
- Setting True Breakpoints and Bound Unit Breakpoints, 8-10

### Break

- Debugger and Break Key Functionality, 7-6, 8-10

### Breakpoint

- Clear All Bound Unit Breakpoints, 8-18
- Clear All Quick Breakpoints, 8-19
- Clear All True Breakpoints, 8-20
- Clear Bound Unit Breakpoint, 8-21
- Clear Quick Breakpoint, 8-22
- Clear True Breakpoint, 8-23
- Guidelines for Setting Breakpoints, 8-12
- List All Bound Unit Breakpoints, 8-39
- List All Quick Breakpoints, 8-40
- List All True Breakpoints, 8-41
- List Bound Unit Breakpoint Directive, 8-42
- List Quick Breakpoint, 8-43
- List True Breakpoint, 8-44
- Preliminary Steps for Using Quick Breakpoints, 8-11
- Set Bound Unit Breakpoint, 8-55, 8-56

### Breakpoint (cont)

- Set Quick Breakpoint, 8-58
- Setting Breakpoints, 7-7
- Setting Global Breakpoints, 8-11
- Setting Quick Breakpoints, 8-11
- Setting True Breakpoints and Bound Unit Breakpoints, 8-10
- Set True Breakpoint, 8-61, 8-62

### Buffer

- Adding Lines to the Current Buffer, 5-123
- Auxiliary Buffer Directives and Escape Sequences, 5-65
- Buffer Status (X), 5-68
- Buffer Status, 5-68, 5-129
- Change Buffer (Bx), 5-70
- Current and Auxiliary Buffers, 5-125
- Deleting All Lines in Current Buffer, 5-118
- Deleting Lines in Current Buffer, 5-118
- Saving Modified Buffer Contents, 5-130

### Call

- Call Cancel, 6-21

### Chaining

- Message Chaining, 11-10

### Change

- Change, 5-27, 7-12
- Change (Change or C), 4-17
- Change All (Change All or Ca), 4-19
- Change Block (Change Block or Cb), 4-21
- Change Buffer (Bx), 5-70
- Change Memory, 8-16
- Change Origin of Text
- Scroll Change (Scroll Change or Sc), 4-30

### Changing

- Changing All Occurrences of a String, 5-121

## INDEX

- Changing (cont)
  - Changing an Existing Source Unit, 4-7, 5-22
  - Changing Character Strings Within a Line, 5-120
  - Changing Line Contents, 5-120
  - Changing Your Working Directory, 3-16
- Characters
  - Special Characters, 4-13
  - Use of Escape Characters, 5-115
- Character String
  - Changing Character Strings Within a Line, 5-120
  - Character String Addressing, 5-113
  - Deleting Character Strings, 5-122
- Clear
  - Clear Abnormal Trap Bit, 8-17
  - Clear All Bound Unit Breakpoints, 8-18
  - Clear All Quick Breakpoints, 8-19
  - Clear All True Breakpoints, 8-20
  - Clear Bound Unit Breakpoint, 8-21
  - Clear Quick Breakpoint, 8-22
  - Clear System Bit, 10-8
  - Clear True Breakpoint, 8-23
- Command
  - Command Level, 3-14
  - Using System Commands in the Editor, 5-130
- Compiling
  - Compiling a Program for Use With the Debugger, 7-26
- Compound
  - Compound Addresses, 5-11
- Connecting
  - Connecting a User to the Executive, 2-2
  - Connecting the Terminal to the Central Processor, 2-1
- Control
  - Directory Control, 3-16
  - File Control, 3-19
  - Printing Control, 3-25
  - Taking a Dump Using a Control Panel, 9-8
  - Taking a Dump Using the System Control Facility (SCF), 9-9
  - Volume Control, 3-15
- Copy
  - Copy (K), 5-76
  - Copy Block, 4-51
- Copy-Append
  - Copy-Append (!k), 5-78
  - Copy-Append, 5-79
- Copying
  - Copying Files, 3-21
- Creating
  - Creating a Dump Volume, 9-3
  - Creating a File, 5-110
  - Creating a Source Unit, 4-7, 5-21
  - Creating Directories, 3-17
  - Creating Files, 3-19
  - Creating Volumes, 3-15
  - Creating Work Files, 5-108
  - Sample Screen for Creating a File (Fig), 4-5
- Current
  - Adding Lines to the Current Buffer, 5-123
  - Current and Auxiliary Buffers, 5-125
  - Deleting Lines in Current Buffer, 5-118
  - New Current Line (N), 5-58
  - Use of Period (.) for Current Line, 5-113

## INDEX

- Cursor
  - Cursor Down, 4-70
  - Cursor Left, 4-71
  - Cursor Right, 4-72
  - Cursor Up, 4-73
- Debugger
  - Debugger and Break Key
    - Functionality, 7-6, 8-10
  - Debugger Directives, 7-8, 8-13
  - Debugger File Requirements, 8-3
  - Debugger Memory
    - Requirements, 8-3
  - Debugger Operation, 8-3
  - Executing Your Program With the Debugger, 7-30
  - Invoking the Debugger (Numeric Mode), 8-2
  - Invoking the Debugger (Symbolic Mode), 7-2
  - Invoking the Debugger, 7-28
  - Linking an Object Unit With the Debugger, 7-27
  - Multi-user Debugger (Numeric Mode) Procedures, 8-70
  - Multiuser Debugger (Symbolic Mode) Procedures, 7-26
- Debugging
  - Debugging Multiple Bound Units, 7-30
  - Line Editor Debugging Directives, 5-85
- Deferred
  - Deferred Printing, 3-25
- Define
  - Define Directive Line, 8-27
  - Define Trace, 8-28
- Delete
  - Delete (D), 5-35
  - Delete, 5-36
  - Delete Block, 4-53
  - Global Delete, 5-124
- Deleting
  - Adding and Deleting Lines, 5-120
  - Deleting All Lines in Current Buffer, 5-118
  - Deleting Character Strings, 5-122
  - Deleting Directories, 3-19
  - Deleting Files, 3-21
  - Deleting Lines in Current Buffer, 5-118
  - Deleting Multiple Lines, 5-118
- Designator
  - Parameter Designator Format, 11-6
  - Parameter Designators, 11-5
- Dialup
  - Dialup Terminal, 2-2
- Direct
  - Direct Login Terminal, 2-6
- Direct-Connect
  - Direct-Connect Terminal, 2-2
- Directives
  - Auxiliary Buffer Directives and Escape Sequences, 5-65
  - Controlling the Directive File, 6-6
  - Debugger Directives, 7-8, 8-13
  - Define Directive Line, 8-27
  - Directive Region, 4-6
  - Edit Mode Description and Directives, 5-33
  - Entering Directives, 8-4
  - Entering Linker Directives, 6-10
  - Entering Screen Editor Directives, 4-9
  - General Advanced Line Editor Directives, 5-50
  - Global Directives, 5-124
  - Input Mode Description and Directives, 5-22
  - Line Editor Debugging Directives, 5-85

## INDEX

### Directives (cont)

- Line Editor Directive
  - Format Conventions, 5-3
- Line Editor Programming Directives, 5-92
- Linker Directive
  - Categories, 6-3
- Linker Directives, 6-13
- List Bound Unit Breakpoint Directive, 8-42
- Patch Directives, 10-7
- Screen Editor Directive
  - Format Conventions, 4-9
- Screen Editor Directives, 4-15
- Submitting Patch Directives, 10-4
- Summary of Line Editor Directives and Escape Sequence (Tbl), 5-16
- Summary of Line Editor Directives and Escape Sequences, 5-16
- Summary of Numeric Mode Directives (Tbl), 8-5
- Summary of Screen Editor Directives (Tbl), 4-14
- Summary of Screen Editor Directives, 4-13
- Summary of Symbolic Mode Directives (Tbl), 7-3
- Symbols Used in Numeric Mode Directive Lines (Tbl), 8-6
- Terms Used in Symbolic Mode Directives (Tbl), 7-4

### Directory

- Changing Your Working Directory, 3-16
- Creating Directories, 3-17
- Deleting Directories, 3-19
- Directories, 3-2
- Directory Control, 3-16
- Example of Disk File Directory Structure (Fig), 3-2
- Intermediate Directories, 3-3
- Listing Files and Directories, 3-23

### Directory (cont)

- Locations of Disk
  - Directories and Files, 3-5
  - Renaming Directories, 3-19
  - Root Directory, 3-3
  - Sample Directory Listing (Fig), 2-7
  - Sample Directory Structure (Fig), 3-4
  - System Root Directory, 3-3
  - User Root Directories, 3-3
  - Working Directory, 3-4

### Disk

- Determining Available Disk Space, 9-5
- Disk File Conventions, 3-2
- Example of Disk File Directory Structure (Fig), 3-2
- Locations of Disk
  - Directories and Files, 3-5
  - Renaming Disk Volumes, 3-16

### DPEDIT

- DPEDIT Command, 9-17
- Operating Procedure for DPEDIT, 9-19

### Dump

- Creating a Dump Volume, 9-3
- Dump, 7-14
- Dump Memory, 8-30
- Hexadecimal Dump, 5-87
- Interpreting and Using Memory Dumps, 9-19
- Logical Dump, 9-12
- Physical Dump, 9-12
- Shared Dump and System Volumes, 9-7
- Significant Locations On Memory Dump (Tbl), 9-20
- Taking a Dump Using a Control Panel, 9-8
- Taking a Dump Using the System Control Facility (SCF), 9-9
- User Level Active at the Time of Dump, 9-22
- Using the Dump Utilities, 9-2

## INDEX

### Dumpfile

- Dumpfile Format, 9-5
- Maximum Dumpfile Size, 9-5
- Setting Dumpfile Size, 9-4

### Editor

- Advanced Functions of the Line Editor, 5-50
- Entering Screen Editor Directives, 4-9
- General Advanced Line Editor Directives, 5-50
- Initiating a Line Editor Session, 5-108
- Interrupting Screen Editor Processing, 4-8
- Line Editor Debugging Directives, 5-85
- Line Editor Directive Format Conventions, 5-3
- Line Editor Modes, 5-110
- Line Editor Programming Directives, 5-93
- Line Editor Suffix Conventions, 5-3
- Loading the Line Editor, 5-14
- Loading the Screen Editor, 4-3
- Quitting the Line Editor, 5-109
- Screen Editor Directive Format Conventions, 4-9
- Screen Editor Directives, 4-15
- Screen Editor Processing, 4-2
- Screen Editor Suffix Conventions, 4-3
- Screen Editor Template for Microsystem 6/10 Keyboard (Fig), 4-46
- Screen Editor Template for VIP7200 Keyboard (Fig), 4-46
- Screen Editor Template for VIP7201 Keyboard (Fig), 4-46
- Screen Editor Template for VIP7300 Word Processing Keyboard (Fig), 4-46

### Editor (cont)

- Screen Editor Template for VIP730x General Purpose and Data Entry Keyboard (Fig), 4-46
- Screen Editor Template for VIP780x General Purpose Keyboard (Fig), 4-46
- Summary of Line Editor Directives and Escape Sequences, 5-16
- Summary of Screen Editor Directives, 4-13
- Using System Commands in the Editor, 5-130

### Entering

- Entering Directives, 8-4
- Entering Linker Directives, 6-10
- Entering Screen Editor Directives, 4-9

### Erase

- Erase Block, 4-54
- Erase EOL, 4-76

### Executive

- Connecting a User to the Executive, 2-2

### File

- 9-Track Magnetic Tape File Organization, 3-11
- Controlling the Directive File, 6-6
- Copying Files, 3-21
- Creating a File, 5-110
- Creating Files, 3-19
- Creating Work Files, 5-108
- Debugger File Requirements, 8-3
- Deleting Files, 3-21
- Directing Output to a File, 3-24
- Disk File Conventions, 3-2
- File Control, 3-19
- File In, 8-34
- File Out, 8-35
- Listing Files and Directories, 3-23
- Locating Files, 3-22



## INDEX

### File (cont)

- Locations of Disk
  - Directories and Files, 3-5
- Magnetic Tape File and Volume Names, 3-11
- Magnetic Tape File Conventions, 3-11
- Moving Lines in a File, 5-127
- Printing Files at Your Terminal, 3-25
- Reading File Contents, 5-117
- Renaming Files, 3-21
- Repeating Lines in a File, 5-126
- Reserving Files or Devices, 3-28
- Reset File, 8-53
- Sample Screen for Creating a File (Fig), 4-5
- Sample Screen for Modifying a File (Fig), 4-5
- Saving File Contents, 5-117
- Specify File, 8-64, 8-66
- Standard I/O Files, 3-13
- Unit-Record Device File Conventions, 3-12
- Using Existing Files, 5-128
- Working With Files, 3-13

### Form

- Forms Login, 2-4
- Login Arguments Form (Fig), 2-6
- Login Form (Fig), 2-5

### Format

- Dumpfile Format, 9-4
- Line Editor Directive
  - Format Conventions, 5-3
- Link Map Formats (Fig), 6-56
- Message Format, 11-3
- Parameter Designator
  - Format, 11-6
- Screen Editor Directive
  - Format Conventions, 4-9

### Function

- Function Keys, 4-46

### Global

- Global (G), 5-54
- Global, 5-55
- Global Delete, 5-124
- Global Directives, 5-124
- Global Print, 5-125
- Set Global Share Bit Off, 10-35
- Set Global Share Bit On, 10-36
- Setting Global Breakpoints, 8-11

### Group

- Group Libraries, 11-2
- Task Group Structures, 9-15

### Halt

- Halt at Level 2, 9-21

### Header

- Print Header Line, 8-48

### Hexadecimal

- Hexadecimal Dump (ZDUMP), 5-86
- Hexadecimal Dump, 5-87
- Hexadecimal Patch, 10-18
- Print Hexadecimal Value, 8-49

### History

- Maintaining a Trace
  - History, 7-7, 8-13

### I/O

- Standard I/O Files, 3-13

### Input

- Change Origin of Text
  - During Input Mode, 5-75
- Input Mode Description and Directives, 5-22

### Insert

- Insert, 5-30, 5-32
- Inserting Lines, 5-123

### Intermediate

- Intermediate Directories, 3-3

## INDEX

### Interrupting

- Interrupting a Task, 2-9
- Interrupting Execution, 3-22
- Interrupting Linker Execution, 6-93
- Interrupting Screen Editor Processing, 4-8

### J-Mode

- Start J-Mode Trace, 8-67

### Keys

- Correlation of Scorpeo's Labeled Keys (Tbl), 4-64
- Function Keys, 4-46
- Labeled Keys, 4-62

### Left

- Cursor Left, 4-71
- Left Margin (Left Margin or Lm), 4-25
- Window Left, 4-59

### Level

- Command Level, 3-14
- Determining the Active Level, 8-12
- Halt at Level 2, 9-21
- User Level Active at the Time of Dump, 9-21

### Library

- Adding a Message to the Message Library, 11-15
- Message Libraries, 11-2
- Message Library Record Structure (Fig), 11-4
- Message Library Utilities, 11-13
- Standard Messages in the System Message Library, 11-12
- Updating the Message Library, 11-14

### Link

- Link, 6-46
- Link Map Formats (Fig), 6-56
- Producing Link Map(S), 6-5
- Sample Link Sessions, 6-94

### Linker

- Entering Linker Directives, 6-10
- Interrupting Linker Execution, 6-93
- Linker Directive Categories, 6-3
- Linker Directives, 6-13
- Linker Functions, 6-1
- Linker Procedures, 6-93
- Loading the Linker, 6-7
- Sample Linker Dialogs, 7-27
- Terminating the Linker, 6-7

### List

- List, 7-18
- List All Bound Unit Breakpoints, 8-39
- List All Quick Breakpoints, 8-40
- List All True Breakpoints, 8-41
- List Bound Unit Breakpoint Directive, 8-42
- List Group Patch Names, 10-25
- Listing Files and Directories, 3-23
- List Patch Names, 10-30
- List Patches, 10-27
- List Patches Now, 10-29
- List Quick Breakpoint, 8-43
- List Specified Group Patch, 10-26
- List Specified Patch, 10-32
- List True Breakpoint, 8-44
- List Updates, 10-33
- Sample Directory Listing (Fig), 2-7

### Loading

- Loading Patch, 10-3
- Loading the Line Editor, 5-14
- Loading the Linker, 6-7
- Loading the Screen Editor, 4-3

### Locating

- Locating Files, 3-22

## INDEX

- Logical
  - Logical Dump, 9-12
- Login
  - Banner Login, 2-3
  - Direct Login Terminal, 2-6
  - Forms Login, 2-4
  - Login Arguments Form (Fig), 2-6
  - Login Form (Fig), 2-5
  - Login Terminal, 2-2
  - Manual Login Terminal, 2-3
- Magnetic Tape
  - 9-Track Magnetic Tape File Organization, 3-11
  - Magnetic Tape Device Pathname Construction, 3-12
  - Magnetic Tape File and Volume Names, 3-11
  - Magnetic Tape File Conventions, 3-11
- Manual
  - Manual Login Terminal, 2-3
- Map
  - Link Map Formats (Fig), 6-56
  - Producing Link Map(s), 6-5
- Memory
  - Change Memory, 8-16
  - Debugger Memory Requirements, 8-3
  - Display Memory, 8-29
  - Dump Memory, 8-30
  - Get Quick Memory, 8-36
  - Interpreting and Using Memory Dumps, 9-19
  - Memory Pool Structures, 9-15
  - Print Quick Memory Pointer, 8-50
  - Return Quick Memory, 8-54
  - Significant Locations On Memory Dump (Tbl), 9-20
- Message
  - Adding a Message to the Message Library, 11-15
- Message (cont)
  - Message Chaining, 11-10
  - Message Code, 11-4
  - Message Format, 11-3
  - Message Libraries, 11-2
  - Message Library Record Structure (Fig), 11-4
  - Message Library Utilities, 11-13
  - Message Reporter, 11-1
  - Message Structure, 11-14
  - Message Text, 11-5
  - Parameterized Messages, 11-5
  - Sending Messages to the Operator, 2-9
  - Standard Messages in the System Message Library, 11-12
  - System Message Library, 11-2
  - Updating the Message Library, 11-14
- Move
  - Move (M), 5-81
  - Move, 5-81
  - Move Block, 4-56
- Move-Append
  - Move-Append (!m), 5-83
  - Move-Append, 5-83
- Multiuser Debugger
  - Multiuser Debugger (Numeric Mode) Procedures, 8-70
  - Multiuser Debugger (Symbolic Mode) Procedures, 7-26
- Non-Login
  - Non-Login Terminal, 2-6
- Numeric Mode
  - Invoking the Debugger (Numeric Mode), 8-2
  - Multiuser Debugger (Numeric Mode) Procedures, 8-70
  - Summary of Numeric Mode Directives (Tbl), 8-5

## INDEX

- Operator
  - Sending Messages to the Operator, 2-9
- Output
  - Directing Output to a File, 3-24
  - Directing Output to a Printer, 3-24
- Overlay
  - Creating a Root and Optional Overlay(s), 6-4
  - Using Overlays, 6-93
  - Overlaytable, 6-66
- Parameterized
  - Parameterized Messages, 11-5
- Patch
  - Applying the Patch, 10-7
  - Data Patch, 10-10
  - Eliminate Patch, 10-14
  - Hexadecimal Patch, 10-18
  - List Group Patch Names, 10-25
  - List Patch Names, 10-30
  - List Patches, 10-27
  - List Patches Now, 10-29
  - List Specified Group Patch, 10-25
  - List Specified Patch, 10-32
  - Loading Patch, 10-3
  - Naming the Patch, 10-7
  - Patch Directives, 10-7
  - Submitting Patch Directives, 10-4
  - Symbolic Data Patch, 10-40
  - Symbolic Patch, 10-43
- Pathname
  - Absolute and Relative Pathnames, 3-8
  - Magnetic Tape Device Pathname Construction, 3-12
  - Sample Pathnames (Fig), 3-9
  - Symbols Used in Pathnames, 3-6
- Physical
  - Physical Dump, 9-12
- Pool
  - Memory Pool Structures, 9-15
- Primary
  - Primary Libraries, 11-3
- Print
  - Global Print, 5-125
  - Print (P), 5-37
  - Print, 5-38, 5-40, 8-46
  - Print All, 8-47
  - Print Header Line, 8-48
  - Print Hexadecimal Value, 8-49
  - Print Line Number (=!/p), 5-59
  - Print Line Number, 5-59
  - Print Quick Memory Pointer, 8-50
  - Print Trace, 8-51
  - Print With Line Number (!p), 5-61
  - Print With Line Number, 5-61
- Printer
  - Directing Output to a Printer, 3-23
  - Writing to Line Printer, 5-130
- Printing
  - Deferred Printing, 3-25
  - Printing Control, 3-25
  - Printing Files at Your Terminal, 3-25
  - Printing Line Numbers, 5-112
- Quick
  - Clear All Quick Breakpoints, 8-19
  - Clear Quick Breakpoint, 8-22
  - Get Quick Memory, 8-36
  - List All Quick Breakpoints, 8-40
  - List Quick Breakpoint, 8-43

## INDEX

- Quick (cont)
  - Preliminary Steps for Using Quick Breakpoints, 8-11
  - Print Quick Memory Pointer, 8-50
  - Return Quick Memory, 8-54
  - Set Quick Breakpoint, 8-58
  - Setting Quick Breakpoints, 8-11
- Region
  - Directive Region, 4-6
  - Status Region, 4-6
  - Text Region, 4-6
- Relative
  - Absolute and Relative Pathnames, 3-7
- Reserving
  - Reserving Files or Devices, 3-28
- Right
  - Cursor Right, 4-72
  - Right Margin (Right Margin or Rm), 4-29
  - Window Right, 4-60
- Root
  - Creating a Root and Optional Overlay(S), 6-4
  - Root Directory, 3-3
  - System Root Directory, 3-3
  - User Root Directories, 3-3
- SCF
  - Taking a Dump Using the System Control Facility (SCF), 9-9
- Screen Editor
  - Description of the Screen, 4-5
  - Entering Screen Editor Directives, 4-9
  - Interrupting Screen Editor Processing, 4-8
  - Loading the Screen Editor, 4-3
  - Screen Editor Directive Format Conventions, 4-9
- Screen Editor (cont)
  - Screen Editor Directives, 4-15
  - Screen Editor Processing, 4-2
  - Screen Editor Suffix Conventions, 4-3
  - Screen Editor Template for Microsystem 6/10 Keyboard (Fig), 4-46
  - Screen Editor Template for VIP7200 Keyboard (Fig), 4-46
  - Screen Editor Template for VIP7201 Keyboard (Fig), 4-46
  - Screen Editor Template for VIP7300 Word Processing Keyboard (Fig), 4-46
  - Screen Editor Template for VIP730x General Purpose and Data Entry Keyboard (Fig), 4-46
  - Screen Editor Template for VIP780x General Purpose Keyboard (Fig), 4-46
  - Summary of Screen Editor Directives (Tbl), 4-14
  - Summary of Screen Editor Directives, 4-13
- Set
  - Guidelines for Setting Breakpoints, 8-12
  - Set, 7-22
  - Set Bound Unit Breakpoint, 8-55, 8-56
  - Set Global Share Bit Off, 10-35
  - Set Global Share Bit On, 10-36
  - Set Quick Breakpoint, 8-58
  - Set Share Bit Off, 10-37
  - Set Share Bit On, 10-38
  - Set System Bit On, 10-39
  - Setting Breakpoints, 7-7
  - Setting Dumpfile Size, 9-4
  - Setting Global Breakpoints, 8-11
  - Setting Quick Breakpoints, 8-11

## INDEX

### Set (cont)

- Setting True Breakpoints and Bound Unit Breakpoints, 8-10
- Set True Breakpoint, 8-61, 8-62,
- Tab Set, 4-84

### Share

- Set Global Share Bit Off, 10-35
- Set Global Share Bit On, 10-36
- Set Share Bit Off, 10-37
- Set Share Bit On, 10-38
- Share, 6-82

### Subsystem Switcher

- Exiting the Subsystem Switcher, 2-8
- Procedures Under the Subsystem Switcher, 2-8
- Selecting a Subsystem, 2-8
- Switching Subsystems, 2-8

### Suffix

- Line Editor Suffix Conventions, 5-3
- Screen Editor Suffix Conventions, 4-3

### Swappool, 6-85

### Symbolic

- Invoking the Debugger (Symbolic Mode), 7-2
- Multiuser Debugger (Symbolic Mode) Procedures, 7-26
- Summary of Symbolic Mode Directives (Tbl), 7-3
- Symbolic Data Patch, 10-40
- Symbolic Mode Special Symbols (Tbl), 7-5
- Symbolic Patch, 10-43
- Terms Used in Symbolic Mode Directives (Tbl), 7-4

### System

- Clear System Bit, 10-8
- Set System Bit On, 10-39

### System (cont)

- Shared Dump and System Volumes, 9-7
- Standard Messages in the System Message Library, 11-12
- System Facilities, 1-1
- System Message Library, 11-2
- System Root Directory, 3-3
- System Summary, 9-12
- Taking a Dump Using the System Control Facility (SCF), 9-9
- Using System Commands in the Editor, 5-130

### Tab

- Tab Clr, 4-83
- Tab Set, 4-84

### Task

- Interrupting a Task, 2-9
- Task Group Structures, 9-15
- Task Structures, 9-16

### Terminal

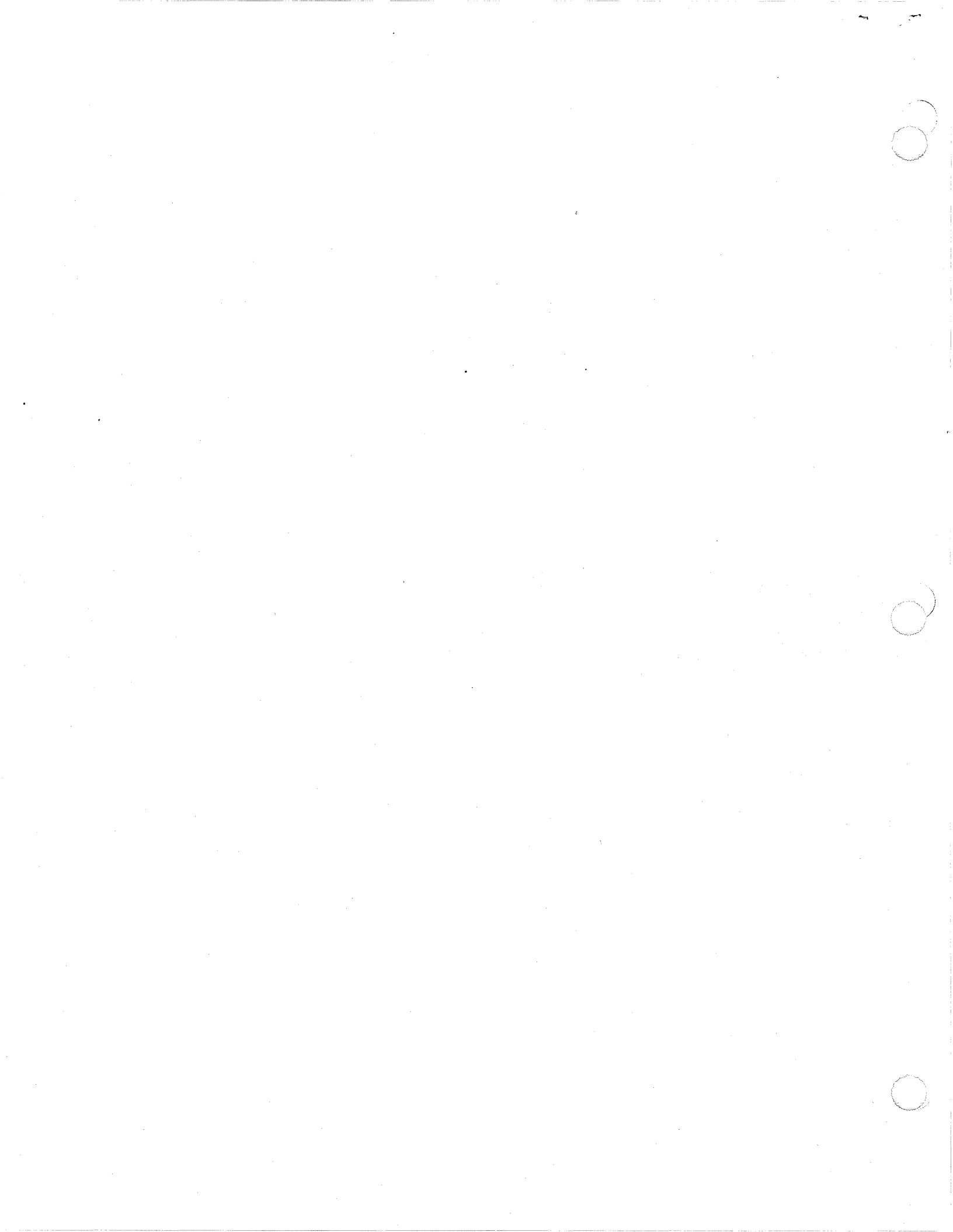
- Accept Single Line From a Terminal (!r), 5-67
- Connecting the Terminal to the Central Processor, 2-1
- Dialup Terminal, 2-2
- Direct Login Terminal, 2-6
- Direct-Connect Terminal, 2-2
- Login Terminal, 2-2
- Manual Login Terminal, 2-3
- Non-Login Terminal, 2-6
- Printing Files at Your Terminal, 3-25
- Redirecting Output to Your Terminal, 3-25

### Trace

- Define Trace, 8-28
- End Trace, 8-31
- Maintaining a Trace History, 7-7, 8-13
- Print Trace, 8-51
- Start J-Mode Trace, 8-67
- Trace, 7-24

## INDEX

- Trap
  - Clear Abnormal Trap Bit, 8-17
  - Turn On Abnormal Trap Bit, 8-68
- Unit-Record
  - Unit-Record Device File Conventions, 3-12
- Up
  - Cursor Up, 4-73
  - Window Up, 4-61
- Utility
  - Using the Dump Utilities, 9-2
  - ZXBTMG Utility, 11-13
  - ZXDMSG Utility, 11-13
- Volume
  - Automatic Tape Volume Recognition, 3-12
  - Creating a Dump Volume, 9-3
  - Creating Volumes, 3-15
  - Magnetic Tape File and Volume Names, 3-11
  - Renaming Disk Volumes, 3-16
  - Shared Dump and System Volumes, 9-7
  - Volume Control, 3-15
- Window
  - Window Down, 4-58
  - Window Left, 4-59
  - Window Right, 4-60
  - Window Up, 4-61
  - Window Width (Window Width or Ww), 4-41
- Work
  - Creating Work Files, 5-108
  - Changing Your Working Directory, 3-16
  - Working Directory, 3-4
  - Working With Files, 3-13
- XRAY
  - Operating Procedure for XRAY, 9-27
  - XRAY Command, 9-24
- ZXBTMG
  - ZXBYMG Utility, 11-13
- ZXDMSG
  - Zxdsmg Utility, 11-13





**HONEYWELL INFORMATION SYSTEMS**  
**Technical Publications Remarks Form**

TITLE

DPS 6  
GCOS 6 MOD 400 APPLICATION DEVELOPER'S GUIDE  
ADDENDUM B

ORDER NO.

CZ15-02B

DATED

MARCH 1987

**ERRORS IN PUBLICATION**

[Empty box for reporting errors in publication]

**SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION**

[Empty box for providing suggestions for improvement to publication]



Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here.

FROM: NAME \_\_\_\_\_

DATE \_\_\_\_\_

TITLE \_\_\_\_\_

COMPANY \_\_\_\_\_

ADDRESS \_\_\_\_\_

\_\_\_\_\_

CUT ALONG LINE

PLEASE FOLD AND TAPE—  
NOTE: U. S. Postal Service will not deliver stapled forms



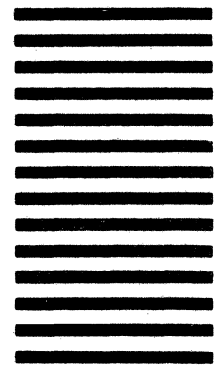
NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

**BUSINESS REPLY MAIL**  
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

**HONEYWELL INFORMATION SYSTEMS**  
200 SMITH STREET  
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486



CUT ALONG

FOLD ALONG LINE

FOLD ALONG LINE

**Honeywell**

**HONEYWELL INFORMATION SYSTEMS**  
**Technical Publications Remarks Form**

CUT ALONG LINE

TITLE

DPS 6  
GCOS 6 MOD 400 APPLICATION DEVELOPER'S GUIDE  
ADDENDUM B

ORDER NO.

CZ15-02B

DATED

MARCH 1987

**ERRORS IN PUBLICATION**

[Empty box for reporting errors in publication]

**SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION**

[Empty box for providing suggestions for improvement to publication]



Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here.

FROM: NAME \_\_\_\_\_

DATE \_\_\_\_\_

TITLE \_\_\_\_\_

COMPANY \_\_\_\_\_

ADDRESS \_\_\_\_\_

\_\_\_\_\_

PLEASE FOLD AND TAPE—  
NOTE: U. S. Postal Service will not deliver stapled forms

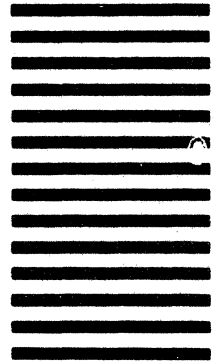


NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

**BUSINESS REPLY MAIL**  
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA 02154

POSTAGE WILL BE PAID BY ADDRESSEE

**HONEYWELL INFORMATION SYSTEMS**  
200 SMITH STREET  
WALTHAM, MA 02154



ATTN: PUBLICATIONS, MS486

**Honeywell**

CUT ALONG  
FOLD ALONG LINE  
FOLD ALONG LINE



**Together. we can find the answers.**

# **Honeywell**

**Honeywell Information Systems**

**U.S.A.:** 200 Smith St., MS 486, Waltham, MA 02154

**Canada:** 155 Gordon Baker Rd., Willowdale, ON M2H 3N7

**U.K.:** Great West Rd., Brentford, Middlesex TW8 9DH **Italy:** 32 Via Pirelli, 20124 Milano

**Mexico:** Avenida Nuevo Leon 250, Mexico 11, D.F. **Japan:** 2-2 Kanda Jimbo-cho Chiyoda-ku, Tokyo

**Australia:** 124 Walker St., North Sydney, N.S.W. 2060 **S.E. Asia:** Mandarin Plaza, Tsimshatsui East, H.K.

45022, 0486, Printed in U.S.A.

CZ15-02