60481400

CYBER INTERACTIVE DEBUG VERSION 1 REFERENCE MANUAL

CDC® OPERATING SYSTEMS:

NOS 1

NOS 2

NOS/BE 1

LANGUAGE-INDEPENDENT CID COMMANDS

Command Format	Short Form	Page
CLEAR,AUXILIARY	CAUX	5-1
CLEAR,BREAKPOINT[,breakpoint-list]	CB	5-1
CLEAR,GROUP[,group-list]	CG	5-1
CLEAR,INTERPRET	CI	5-2
CLEAR,OUTPUT	COUT	5-2
CLEAR,TRAP,type[,trap-list]	CT	5-2
CLEAR,VETO	CV	5-2
DISPLAY,location [,format ,format,count]	D	5-12
ENTER,value,location[,count]	E	5-13
EXECUTE[,location]	EXEC	5-14
GO[,location]	†	5-14
HELP [,* ,subject ,command-name]	†	5-14
JUMP,label	†	5-15
LABEL,label	†	5-15
LIST,BREAKPOINT[,breakpoint-list]	LB	5-3
LIST,GROUP[,group-list]	LG	5-3
LIST,MAP[,place-list]	LM	5-5
LIST,STATUS	LS	5-5
LIST,TRAP,type[,trap-list]	LT	5-5
LIST,VALUES[,place-list]	LV	5-5
MESSAGE,'message text'	†	5-15
MOVE,source,destination[,count]	M	5-15
NULL	††	5-16
PAUSE[,'message text']	†	5-16
QUIT [,NORMAL ,N ,ABORT ,A]	†	5-17
READ {,file-name ,group-name}	†	5-17
SAVE,BREAKPOINT,file-name[,breakpoint-list]	SAVEB	5-7
SAVE,GROUP,file-name,group-list	SAVEG	5-7
SAVE,TRAP,file-name,type[,trap-list]	SAVET	5-7
SAVE,*,file-name	†	5-7
SET,AUXILIARY,file-name[,option-list]	SAUX	5-8
SET,BREAKPOINT,location,[first], [last],[step][[]	SB	5-9
SET,GROUP,group-name[[]	SG	5-9
SET,HOME[,(p,s)],[P.]progname	SH	5-9
SET,INTERPRET {,ON ,OFF}	SI	5-10
SET,OUTPUT[,option-list]	SOUT	5-10
SET,TRAP,type,scope[,report-level][[]	ST	5-11
SET,VETO [,ON ,OFF]	SV	5-12
SKIPIF,value$_1$,relation,value$_2$	†	5-17
STEP,[n],[type],[scope]	S	5-18
SUSPEND[,file-name]	†	5-19
TRACEBACK [,E.entrypoint ,P.progname]	†	5-19

†This command has no short form.
††The short form for the NULL command is an empty line (or an empty area between semicolons).

(BASIC, COBOL, and FORTRAN CID commands are shown on the inside back cover.)

60481400

CYBER INTERACTIVE DEBUG VERSION 1 REFERENCE MANUAL

CDC® OPERATING SYSTEMS:

NOS 1

NOS 2

NOS/BE 1

REVISION RECORD

Revision	Description
A (04-28-78)	Original release.
B (07-20-79)	Revised to reflect CYBER Interactive Debug Version 1.1. The changes include support of BASIC 3 and FORTRAN 5, as well as miscellaneous technical changes and corrections.
C (11-20-81)	Revised to reflect CYBER Interactive Debug Version 1.2 at PSR level 552. The changes include the addition of COBOL 5 features, the addition of the STEP command, and reorganization of the manual, as well as miscellaneous technical changes and corrections. This is a complete reprint.
D (06-25-84)	Revised at PSR level 601. This revision includes release under NOS 2, removal of the time limit condition, clarification of the #ERRCODE variable description, and miscellaneous technical and editorial changes.

REVISION LETTERS I, O, Q, AND X ARE NOT USED

©COPYRIGHT CONTROL DATA CORPORATION
1978, 1979, 1981, 1984
All Rights Reserved
Printed in the United States of America

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
P.O. BOX 3492
SUNNYVALE, CALIFORNIA 94088-3492

or use Comment Sheet in the back of this manual

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

Page	Revision
Front Cover	-
Inside Front Cover	C
Title Page	-
ii	D
iii/iv	D
v	D
vi	D
vii	D
viii	D
ix/x	C
xi	D
1-1	C
1-2	C
2-1 thru 2-3	C
3-1	C
3-2	D
3-3	D
3-4 thru 3-8	C
4-1	D
4-2	D
4-3 thru 4-5	C
4-6	D
4-7 thru 4-9	C
5-1 thru 5-9	C
5-10 thru 5-12	D
5-12.1/5-12.2	D
5-13 thru 5-19	C
6-1 thru 6-6	C
7-1	C
7-2	D
7-3 thru 7-5	C
8-1 thru 8-17	C
A-1	C
A-2	C
B-1 thru B-3	C
B-4	D
B-5	D
B-6	C
B-7	D
B-8	C
B-9	C
C-1	C
D-1	C
E-1	C
F-1	D
F-2	C
G-1	C
Index-1	C
Index-2	C
Index-3	D
Comment Sheet/Mailer	D
Inside Back Cover	C
Back Cover	-

PREFACE

This manual describes the features of the CONTROL DATA® CYBER Interactive Debug facility, a supervisory program designed to aid users in the debugging of object programs (see Introduction, section 1).

CYBER Interactive Debug (CID) operates under the following operating systems:

- NOS 1 for the CDC® CYBER 170 Computer Systems, CYBER 70 Models 71, 72, 73, 74; and 6000 Computer Systems
- NOS 2 for the CDC CYBER 180 Computer Systems; CYBER 170 Computer Systems; CYBER 70 Models 71, 72, 73, 74; and 6000 Computer Systems
- NOS/BE 1 for the CONTROL DATA CYBER 170 Computer Systems; CYBER 70 Models 71, 72, 73, 74; and 6000 Computer Systems

Although intended primarily to be used interactively from a remote time-sharing terminal, CID can be used with limited features in batch mode.

This manual is written for the experienced programmer who knows the programming language of the compiler or assembler used to produce the program to be debugged. It is assumed that you are familiar with the operating system and computer system in use and any terminal employed with CID.

This manual is organized to ease referencing details about CID commands and concepts:

- Section 1 summarizes CID operation briefly.
- Section 2 describes methods used to access CID.
- Section 3 describes in detail concepts that would not be appropriately described under CID command names.
- Sections 4 through 6 describe the syntax and functions of the CID commands.
- Section 7 describes your responses when CID command execution is interrupted as a result of errors, warnings, veto mode prompts, or terminal interrupts.
- Section 8 contains sample debug sessions under CID.

Beginning CID users should read a user's guide for CID before reading this manual.

Related material is contained in the publications listed below; the publications are listed within groupings that indicate relative importance to readers of this manual.

The Software Publications Release History serves as a guide in determining which revision level of software documentation corresponds to the Programming Systems Report (PSR) level of installed site software.

The following publications are of primary interest:

Publication	Publication Number
CYBER Interactive Debug Version 1 Online Reference Manual	L60481400
CYBER Interactive Debug Version 1 Guide for Users of FORTRAN Extended Version 4	60482700
CYBER Interactive Debug Version 1 Guide for Users of FORTRAN Version 5	60484100
INTERCOM Version 5 Reference Manual	60455010
Network Products Interactive Facility Version 1 Reference Manual	60455250
NOS/BE Version 1 Reference Manual	60493800
NOS Version 1 Reference Manual Volume 1 of 2	60435400
NOS Version 2 Reference Manual Volume 1 of 4	60459660

The following publications are of secondary interest:

Publication	Publication Number
BASIC Version 3 Reference Manual	19983900
COBOL Version 5 Reference Manual	60497100
COBOL Version 5 Online Reference Manual	L60497100
COMPASS Version 3 Reference Manual	60492600
FORTRAN Extended Version 4 Reference Manual	60497800
FORTRAN Version 5 Reference Manual	60481300
FORTRAN Version 5 Online Reference Manual	L60481300
Network Products Interactive Facility Version 1 User's Guide	60455260
Network Terminal User's Instant	60455270
Software Publications Release History	60481000

CDC manuals can be ordered from Control Data Corporation, Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103.

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.

CONTENTS

NOTATIONS xi

1. INTRODUCTION 1-1

Debug Session 1-1
CID Features 1-1

2. ACCESSING CID 2-1

Debug Control Statement 2-1
Compilation of High-level Language Programs 2-1
  Compilation of BASIC Programs 2-1
  Compilation of COBOL Programs 2-1
  Compilation of FORTRAN Programs 2-2
Execution Under CID 2-2
Files Used During a Debug Session 2-2
  Input File 2-2
  Output Files 2-2
  CID Environment Files 2-3
  Suspend File 2-3
  Scratch Files 2-3

3. CID CONCEPTS 3-1

Types of CID Commands 3-1
Home Program 3-1
Breakpoints 3-1
Traps 3-2
  Trap Types 3-2
    ABORT Trap 3-2
    END Trap 3-2
    FETCH Trap 3-3
    INSTRUCTION Trap 3-3
    INTERRUPT Trap 3-3
    JUMP Trap 3-3
    LINE Trap 3-3
    OVERLAY Trap 3-3
    PROCEDURE Trap 3-4
    RJ Trap 3-4
    STORE Trap 3-4
    XJ Trap 3-4
  Default Traps 3-4
  Trap Action 3-4
    Trap Action for BASIC Programs 3-5
    Trap Action for COBOL Programs 3-5
    Trap Action for FORTRAN Programs 3-6
    Trap Action for Other Programs 3-6
  Interpret Mode 3-6
Command Sequences 3-7
  Breakpoint and Trap Bodies 3-7
  Groups 3-7
  Line Sequences 3-7
  Collect Mode 3-7
  CID Sequence Execution 3-7
  Command Sequence Files 3-7
  Command Sequence Examples 3-7
  Editing a Command Sequence 3-7

4. SYNTAX OF LANGUAGE-INDEPENDENT COMMANDS 4-1

Format of Language-Independent Commands 4-1
Addresses 4-1
  Absolute Addresses 4-1
  Module Relative Addresses 4-1
  Entry Point Addresses 4-2
  Overlay Addresses 4-2
  Source Language Symbol Addresses 4-3
    BASIC Symbols 4-3
    COBOL Symbols 4-3
    FORTRAN Symbols 4-3
  Address Range Specification 4-4
  Module or Block Referencing 4-4
  Ellipsis 4-4
Values 4-4
  Numeric Constant Values 4-4
    Decimal Integer Constants 4-4
    Octal Integer Constants 4-5
  Address Values 4-5
  Debug Variables 4-5
    Debug User Variables 4-5
    Debug State Variables 4-5
    Program State Variables 4-5
    Interpret Mode Variables 4-6
    Register State Variables 4-6
Expressions 4-6
  Addition and Subtraction Operators 4-8
  Value Operator (!) 4-8

5. LANGUAGE-INDEPENDENT COMMANDS 5-1

CLEAR Commands 5-1
  CLEAR,AUXILIARY Command 5-1
  CLEAR,BREAKPOINT Command 5-1
  CLEAR,GROUP Command 5-1
  CLEAR,INTERPRET Command 5-2
  CLEAR,OUTPUT Command 5-2
  CLEAR,TRAP Command 5-2
  CLEAR,VETO Command 5-2
LIST Commands 5-3
  LIST,BREAKPOINT Command 5-3
  LIST,GROUP Command 5-3
  LIST,MAP Command 5-5
  LIST,STATUS Command 5-5
  LIST,TRAP Command 5-5
  LIST,VALUES Command 5-5
    BASIC Program Modules 5-6
    COBOL Program Modules 5-6
    FORTRAN Program Modules 5-6
SAVE Commands 5-7
  SAVE,BREAKPOINT Command 5-7
  SAVE,GROUP Command 5-7
  SAVE,TRAP Command 5-7
  SAVE,* COMMAND 5-7
SET Commands 5-7
  SET,AUXILIARY Command 5-8
  SET,BREAKPOINT Command 5-9
  SET,GROUP Command 5-9
  SET,HOME Command 5-9

SET,INTERPRET Command 5-10
SET,OUTPUT Command 5-11
SET,TRAP Command 5-11
SET,VETO Command 5-12.1/ 5-12.2

Other Language-Independent Commands 5-12.1/ 5-12.2

DISPLAY Command 5-12.1/ 5-12.2
ENTER Command 5-13
EXECUTE Command 5-14
GO Command 5-14
HELP Command 5-14
JUMP Command 5-15
LABEL Command 5-15
MESSAGE Command 5-15
MOVE Command 5-15
NULL Command 5-16
PAUSE Command 5-16
QUIT Command 5-17
READ Command 5-17
SKIPIF Command 5-17
STEP Command 5-18
SUSPEND Command 5-19
TRACEBACK Command 5-19

6. LANGUAGE-DEPENDENT COMMANDS 6-1

BASIC CID Commands 6-1
GOTO Command 6-1
IF Command 6-1
LET Command 6-2
MAT PRINT Command 6-2
PRINT Command 6-2
COBOL CID Commands 6-2
DISPLAY Command 6-3
GO TO Command 6-3
MOVE Command 6-4
SET Command 6-4
Format 1 SET Command 6-4
Format 2 SET Command 6-5
FORTRAN CID Commands 6-5
Assignment Command 6-5
GOTO Command 6-6
IF Command 6-6
PRINT Command 6-6

7. ERROR, WARNING, VETO MODE, AND INTERRUPT PROCESSING 7-1

Error Processing 7-1
Warning Processing 7-1
Veto Mode Processing 7-1
Interrupt Processing 7-1
Error, Warning, Veto Mode and Interrupt Responses 7-2
Error Responses 7-2
Warning Responses 7-2
Veto Mode Responses 7-2
Interrupt Responses 7-2

8. SAMPLE DEBUG SESSIONS 8-1

APPENDIXES

A Standard Character Sets A-1
B Diagnostics B-1
C Glossary C-1
D Program Structure D-1
E Batch CID Features E-1
F Usage Constraints and Dependencies F-1
G Debugging Hints G-1

INDEX

FIGURES

2-1 DEBUG Control Statement 2-1
3-1 Trap Report Message 3-4
3-2 CID Sequence Example 3-8
3-3 Defined Group 3-8
4-1 Decimal Integer Constant 4-5
4-2 Octal Integer Constant 4-5
4-3 Abort Information Variables 4-6
4-4 Interpret Mode Variables 4-7
5-1 CLEAR,AUXILIARY Command 5-1
5-2 CLEAR,BREAKPOINT Command 5-2
5-3 CLEAR,GROUP Command 5-2
5-4 CLEAR,INTERPRET Command 5-2
5-5 CLEAR,OUTPUT Command 5-2
5-6 CLEAR,TRAP Command 5-3
5-7 CLEAR,VETO Command 5-3
5-8 LIST,BREAKPOINT Command 5-4
5-9 LIST,GROUP Command 5-4
5-10 LIST,MAP Command 5-5
5-11 LIST,STATUS Command 5-5
5-12 LIST,TRAP Command 5-6
5-13 LIST,VALUES Command 5-6
5-14 SAVE,BREAKPOINT Command 5-7
5-15 SAVE,GROUP Command 5-8
5-16 SAVE,TRAP Command 5-8
5-17 SAVE,* Command 5-8
5-18 SET,AUXILIARY Command 5-9
5-19 SET,BREAKPOINT Command 5-10
5-20 SET,GROUP Command 5-10
5-21 SET,HOME Command 5-10
5-22 SET,INTERPRET Command 5-11
5-23 SET,OUTPUT Command 5-11
5-24 SET,TRAP Command 5-12
5-25 SET,VETO Command 5-12.1/ 5-12.2
5-26 DISPLAY Command 5-12.1/ 5-12.2
5-27 ENTER Command 5-13
5-28 EXECUTE Command 5-14
5-29 GO Command 5-14
5-30 HELP Command 5-15
5-31 JUMP Command 5-15
5-32 LABEL Command 5-15
5-33 MESSAGE Command 5-15
5-34 MOVE Command 5-16
5-35 NULL Command 5-16
5-36 PAUSE Command 5-17
5-37 QUIT Command 5-17
5-38 READ Command 5-17
5-39 SKIPIF Command 5-18
5-40 STEP Command 5-18
5-41 SUSPEND Command 5-19
5-42 TRACEBACK Command 5-19
6-1 GOTO Command 6-1
6-2 IF Command 6-1
6-3 LET Command 6-2
6-4 MAT PRINT Command 6-2
6-5 PRINT Command 6-2
6-6 DISPLAY Command 6-3
6-7 GO TO Command 6-3
6-8 MOVE Command 6-4
6-9 Format 1 SET Command 6-4

6-10 Format 2 SET Command 6-5
6-11 Assignment Command 6-5
6-12 GOTO Command 6-6
6-13 IF Command 6-6
6-14 PRINT Command 6-6
7-1 Command Sequence for Interrupt Example 7-5
8-1 Session A Source Listing 8-1
8-2 Debug Session A 8-1
8-3 Session A Auxiliary File Listing 8-3
8-4 Session B FORTRAN Main Program and Subroutines 8-4
8-5 Debug Session B 8-6
8-6 Session C Source Listing 8-8
8-7 Debug Session C 8-9
8-8 Session C Auxiliary File Listing 8-10
8-9 Session D; COBOL Main Program 8-11
8-10 Session D; FORTRAN Subroutine 8-14
8-11 Session D; Input Data on File SALES 8-14
8-12 Debug Session D 8-15

TABLES

4-1 Debug State Variables 4-5
4-2 Program State Variables 4-5
4-3 Register State Variables 4-8
5-1 CLEAR Command Variants 5-1
5-2 LIST Command Variants 5-4
5-3 SAVE Command Variants 5-7
5-4 SET Command Variants 5-8
5-5 Trap Types 5-12
5-6 DISPLAY Formats 5-13
6-1 Allowable MOVE Command Sending and Receiving Items 6-4
6-2 Allowed Format 1 Set Command Operations 6-5
7-1 Response Keyword Actions 7-3
7-2 Error Response Examples 7-4
7-3 Warning Response Examples 7-4
7-4 Veto Mode Response Examples 7-4
7-5 Interrupt Response Examples 7-4

NOTATIONS

The following notations appear in formats throughout this manual:

{ } Braces indicate that you must specify exactly one of the enclosed vertically stacked parameters or keywords.

[ ] Square brackets indicate that the enclosed parameters or keywords are optional. When two or more items are stacked vertically within brackets, one or none of the items can be used. In some commands, a single left bracket is a valid character.

_ The underscore in CID indicates a relative address in a program module or common block. The symbol to the left of the underscore specifies a block or module name. The symbol to the right of the underscore specifies the relative address within the block or module.

UPPERCASE Uppercase characters indicate words, acronyms, or mnemonics either required by CID or produced as output by CID.

lowercase Lowercase characters indicate words or values that, if entered by you or produced as output by CID, can take on one of several values.

Unless otherwise specified, numbers in this manual are decimal numbers.

INTRODUCTION 1

CYBER Interactive Debug (CID) Version 1 is a facility designed to help you find errors in a program while the program is executing. You can use CID to debug a single program module, a set of program modules with all required object library modules, or a set of programs contained on overlays.

CID is a supervisory program module loaded in the field length to operate on loaded object programs. Consequently, you can use CID to debug programs produced by any assembler or compiler in the product set. There are some constraints, however, on programs that can be debugged under CID (see appendix F).

No special statements are needed in a source program to be debugged under CID; however, more features are available in debugging BASIC, COBOL, and FORTRAN programs if the programs are compiled for use with CID. Compilation for use with CID is described in section 2.

Although CID is designed for interactive use, CID can be used with batch jobs. Not all of the features available for interactive use are also available for batch use. See appendix E for batch job considerations.

DEBUG SESSION

A debug session is the execution of a program under CID control. To begin a debug session, you must load and execute the program with debug mode turned on. (The DEBUG control statement turns on debug mode.) Loading the program with debug mode turned on causes the CID supervisory module to be loaded when the relocatable object program being debugged is loaded. When the program is executed, CID takes control and issues the following message to indicate that the debug session has begun:

```
CYBER INTERACTIVE DEBUG
```

CID then issues a question mark prompt to indicate that you can enter CID commands.

During a debug session, the program being debugged is executed until user-set conditions (and some default conditions) cause execution to be suspended. While execution is suspended, you can observe and change the values of variables, data items, and memory locations associated with the program being debugged. When you are done changing and observing values, you can resume program execution or terminate the debug session. Execution can be suspended any number of times during a debug session, and conditions causing execution to be suspended can be changed any time during the session. The debug session ends when you enter the QUIT command.

CID FEATURES

CID features are listed below:

- Breakpoints

  You can set breakpoints that suspend program execution when specified places in the program are reached. The SET,BREAKPOINT command sets breakpoints.

- Traps

  You can set traps that suspend program execution when specified conditions occur. The SET,TRAP command sets traps.

- High-level language features

  High-level language features are available only for BASIC, COBOL, and FORTRAN programs compiled for use with CID. You can enter language-dependent CID commands that are similar in syntax and action to statements contained in the source language. In other, language-independent, CID commands, you can specify locations in a high-level form, such as by variable names, line numbers, COBOL procedure names, and FORTRAN statement numbers.

  The STEP command executes a specified number of high-level language program lines or COBOL procedures. Additional traps and additional debug variables are also available for high-level language programs.

- Command sequences

  Sequences of CID commands can be defined as groups, bodies, or line sequences. A group executes when directed to by a READ command. A body executes automatically when a particular trap or breakpoint occurs. A line sequence is a sequence of CID commands contained on one line.

- Debug variables

  Debug variables are provided by CID to give you information such as the current field length, the number of the current source line being executed, and the number of traps or breakpoints currently defined. The language-independent ENTER and DISPLAY commands are entered to change and observe the values of debug variables.

- Suspend and resume features

  When a debug session is taking place, the entire session can be suspended such that control returns to the command mode of the operating system. This feature is most useful when files are to be attached or edited during a session. When the debug session is suspended, breakpoint definitions, trap definitions, group definitions, debug variables, and the current status of the debug session are placed on a local file. The SUSPEND command suspends the entire debug session. The control statement DEBUG(RESUME) causes the suspended debug session to be resumed.

  Commands are also available to save specific breakpoint, trap, and group definitions on a local file. These saved definitions can then be used in future debug sessions. The SAVE commands place the definitions onto a local file. The READ command reads definitions from a local file and makes the definitions available during a debug session.

- Informative output

  Informative output during a debug session includes listings of information such as the breakpoints, traps, or groups currently defined; the values of all program variables or data items; the current status of the debug session; and a load map containing the names of all program modules currently loaded. The LIST commands provide this information.

  Output can be directed to the standard output file (usually the terminal) or to an auxiliary file that you specify. The SET,OUTPUT command specifies the types of output sent to the standard output file, and the SET,AUXILIARY command specifies the name of the auxiliary output file and the types of output sent to the auxiliary output file.

This section describes the actions necessary to execute a program under CID. To execute a program under CID, you must load and execute the program with debug mode turned on. Debug tables and CID-oriented object code must be produced during compilation of BASIC, COBOL, and FORTRAN programs if high-level language CID features are to be available for the debug session.

DEBUG CONTROL STATEMENT

The DEBUG control statement (figure 2-1) turns debug mode on or off or resumes a debug session that was suspended and placed on a file by the SUSPEND command. (The SUSPEND command is described in section 5.)

```
DEBUG     [ ,ON                    ].
          [ ,OFF                   ]
          [ ,RESUME [,file-name]   ]

ON        Default; turns on debug mode.

OFF       Turns off debug mode.

RESUME    Resumes the debug session placed on
          the local file file-name by the
          SUSPEND command. Default for file-
          name is ZZZZZDS.
```

Figure 2-1. DEBUG Control Statement

When a debug session is resumed, the CID environment, the program, and the status of the debug session are read from the file and CID takes control. CID issues the following message when the debug session is resumed:

```
CYBER INTERACTIVE DEBUG RESUMED
```

CID then issues a question mark prompt to indicate that you can enter CID commands.

COMPILATION OF HIGH-LEVEL LANGUAGE PROGRAMS

High-level language features are available for BASIC, COBOL, and FORTRAN programs if the programs are compiled for use with CID. When these programs are compiled for use with CID, debug tables and CID-oriented object code are produced during compilation. High-level language features are not available for programs written in languages other than BASIC, COBOL, or FORTRAN.

COMPILATION OF BASIC PROGRAMS

When a BASIC program is compiled for use with CID, the following features are made available for debugging that program:

- BASIC CID commands (see section 6)
- Source language symbol addresses (see section 4)
- LINE trap type (see section 3)
- #LINE debug variable (see section 4)
- LIST,VALUES command (see section 5)
- STEP command (see section 5)

A BASIC program is compiled for use with CID if the program is compiled with debug mode turned on or if DB=ID is specified in the BASIC control statement. (When debug mode is turned on and DB=0 is specified, the program is not compiled for use with CID.) If debug mode is on when a BASIC program is run using the BASIC subsystem on NOS, the program is compiled for use with CID and the program is executed under CID control.

The debug tables produced during compilation of BASIC programs include symbol tables. The symbol tables enable CID commands to reference variables and line numbers from the BASIC source program. The debug tables also indicate that the program is written in BASIC so that CID can provide the language-dependent BASIC CID commands. The CID-oriented object code identifies the beginning of each BASIC source program line to make the LINE trap type and the STEP command available.

COMPILATION OF COBOL PROGRAMS

When a COBOL program is compiled for use with CID, the following features are made available for debugging that program:

- COBOL CID commands (see section 6)
- Source language symbol addresses (see section 4)
- LINE and PROCEDURE trap types (see section 3)
- #LINE and #PROC debug variables (see section 4)
- LIST,VALUES command (see section 5)
- STEP command (see section 5)

A COBOL program is compiled for use with CID if the program is compiled with debug mode turned on or if DB=ID is specified in the COBOL5 control statement. (When debug mode is on and DB=0 is specified, the program is not compiled for use with CID.)

The debug tables produced during compilation of COBOL programs include symbol tables. The symbol tables enable CID commands to reference data names, line numbers, and procedure names from the COBOL source program. The debug tables also indicate that the program is written in COBOL so that CID can provide the language-dependent COBOL CID commands. The CID-oriented object code identifies the beginning of COBOL source lines and the beginning of COBOL procedures, making the LINE and PROCEDURE trap types and the STEP command available.

COMPILATION OF FORTRAN PROGRAMS

When a FORTRAN program is compiled for use with CID, the following features are made available for debugging that program:

- FORTRAN CID commands (see section 6)
- Source language symbol addresses (see section 4)
- LINE trap type (see section 3)
- #LINE debug variable (see section 4)
- LIST,VALUES command (see section 5)
- STEP command (see section 5)

A FORTRAN program is compiled for use with CID if the program is compiled with debug mode turned on or if the proper option is specified in the FORTRAN compiler call:

- For FORTRAN 5 programs, the DB=ID option in the FTN5 control statement causes the program to be compiled for use with CID. (When debug mode is on and either DB=-ID or DB=0 is specified, the program is not compiled for use with CID.) FORTRAN 5 programs cannot be compiled for use with CID when compiler optimization levels other than OPT=0 are selected. If debug mode is on when a FORTRAN 5 program is run interactively on the FORTRAN subsystem on NOS, the program is compiled for use with CID and the program is executed under CID control.

- For FORTRAN Extended 4 programs, the DB option or the DB=ID option in the FTN control statement causes the program to be compiled for use with CID. (When debug mode is turned on and DB=0 is specified, the program is not compiled for use with CID.)

  When a FORTRAN Extended 4 program is compiled for use with CID, the TS compiler option is selected automatically; the program cannot be compiled for use with CID when optimizing compiler options are selected. If debug mode is on when a FORTRAN program is run interactively on the FTNTS subsystem on NOS, the program is compiled for use with CID and the program is executed under CID control.

The debug tables produced during compilation of FORTRAN programs include symbol tables. The symbol tables enable CID commands to reference variables, line numbers, and statement labels from the FORTRAN source program. The CID-oriented object code identifies the beginning of each FORTRAN source line to make the LINE trap type and the STEP command available.

EXECUTION UNDER CID

Programs compiled for use with CID can be executed with debug mode turned on or off. When a program or set of programs is loaded and executed with debug mode on, the CID supervisory module is loaded automatically with the relocatable object programs being debugged. Execution begins at a location in the CID module; CID issues the following message to indicate that the debug session has begun:

```
CYBER INTERACTIVE DEBUG
```

CID then issues a question mark prompt to indicate that you can enter CID commands. Execution of the program being debugged does not begin until you enter a CID command to begin execution (such as the GO, EXECUTE, or STEP commands). To terminate or suspend a debug session, you can enter the QUIT or SUSPEND commands, respectively.

When a program is loaded and executed with debug mode turned off, the program is not executed under CID control. When a BASIC, COBOL, or FORTRAN program with debug tables and CID-oriented object code is executed without CID control, execution is slightly less efficient than with a program compiled normally; otherwise, execution is the same as with normally compiled programs.

FILES USED DURING A DEBUG SESSION

The following types of files are used during a debug session:

- Input file
- Output files
- CID environment files
- Suspend file
- Scratch files

INPUT FILE

The file used for input during a debug session differs depending on whether the session is executed in interactive or batch mode:

- For interactive debug sessions, input is taken from the file ZZZZZIN. ZZZZZIN is automatically assigned (NOS) or connected (NOS/BE) to the terminal.

- For batch debug sessions, input is taken from the local file DBUGIN (see appendix E).

The input file contains unblocked, zero-byte terminated records.

OUTPUT FILES

You can direct CID output to either the standard output file (determined by CID) or to an auxiliary output file (that you specify). These files contain unblocked, zero-byte terminated records. The output can be directed such that some types of output are sent to the standard output file and other types are sent to the auxiliary file.

The standard output file differs depending on whether the session is executed in interactive or batch mode:

- For interactive debug sessions, the standard output file is ZZZZZOU. ZZZZZOU is automatically assigned (NOS) or connected (NOS/BE) to the terminal.

- For batch debug sessions, the standard output file is DBUGOUT (see appendix E).

The types of output sent to the standard output file are specified in the SET,OUTPUT command (see section 5). You can respecify these types any number of times during the debug session.

The types of output sent to the auxiliary output file and the name of the auxiliary output file are specified using the SET,AUXILIARY command (see section 5). You can respecify these types any number of times during the debut session. The auxiliary file name can also be changed during the debug session; however, only one auxiliary output file is in effect at any given time. The auxiliary output file is a local file.

CID ENVIRONMENT FILES

CID environment files are local files on which trap, breakpoint, and group definitins are saved (as the result of a SAVE command described in section 5). Names of CID environment files are specified when the SAVE command is entered. The files contain zero-byte terminated records; one logical record is written to a file each time a SAVE command specifying that file is executed.

SUSPEND FILE

When a debug session is suspended (by the SUSPEND command described in section 5), the current status of the debug session is saved on a local file specified in the SUSPEND command (the default file name is ZZZZZDS). The suspend file contains multiple binary records and is returned whenever the DEBUG(RESUME) control statement is executed.

SCRATCH FILES

CID uses the following scratch files during a debug session:

- ZZZZZDI

- ZZZZZDO

- ZZZZZDT

- ZZZZZUI

When you use CID to debug a program, you should not have any local files with these names.

It is unlikely that you would find any of these files, except for ZZZZZDT, useful. Debug tables are stored on ZZZZZDT when a BASIC, COBOL, or FORTRAN program is compiled for use with CID. In some cases, it is useful to make this file permanent (see appendix F). ZZZZZDT is a binary file that contains multiple logical records.

This section describes miscellaneous concepts that cannot be appropriately covered under individual command names. The following concepts are described in this section:

- Types of CID commands
- Home program
- Breakpoints
- Traps
- Command Sequences

TYPES OF CID COMMANDS

You can enter the following types of CID commands whenever CID issues a question mark prompt:

- Language-independent commands (see sections 4 and 5) all have similar formats and do not vary in form and usage between different programming languages.
- Language-dependent commands (see section 6) are nearly identical in form and action to statements used in the programming language of the program being debugged. Language-dependent commands are only available for BASIC, COBOL, and FORTRAN programs compiled for use with CID.

HOME PROGRAM

The home program is the program on which you are currently working. In language-independent commands, you can reference locations within the home program more easily than locations outside the home program (see section 4). In language-dependent commands, you can only reference locations within the home program (see section 6).

Whenever CID suspends program execution, the program that was suspended is designated the home program. You can explicitly designate a new home program by entering the SET,HOME command (see section 5).

BREAKPOINTS

A breakpoint is a program location where program execution is to be suspended. Whenever a breakpoint location in a program is reached, you can observe and change program values at the point where program execution is suspended.

The SET,BREAKPOINT command sets a breakpoint, the CLEAR,BREAKPOINT command removes one or more breakpoints, the LIST,BREAKPOINT command lists one or more breakpoints, and the SAVE,BREAKPOINT command saves one or more breakpoints on a local file. See section 5 for descriptions of these commands.

CID assigns each breakpoint a number in the range 1 through 16 when the breakpoint is set. This breakpoint number, referred to in the form #n, provides a convenient way of referring to breakpoints in CID commands. The breakpoint number is also used in the breakpoint report message.

Program execution is suspended whenever execution reaches the breakpoint location and the frequency parameters (specified in the SET,BREAKPOINT command) are met. Execution is suspended before the instructions in the breakpoint location are executed.

If a breakpoint body (see Command Sequences later in this section) is not associated with the breakpoint, a breakpoint report message is automatically issued, and CID issues a question mark prompt to indicate that you can enter CID commands. The form of the breakpoint message is:

```
*B #n, AT location
```

The n is the breakpoint number and location is reported as the breakpoint location specified in the SET,BREAKPOINT command.

If a breakpoint body (see command sequences later in this section) is associated with the breakpoint, the CID commands in the breakpoint body are executed automatically, and you do not gain control unless a PAUSE command is executed in the breakpoint body. (Breakpoint bodies are described later in this section.) If you do not gain control, no breakpoint message is issued.

CID places return jump (RJ) machine instructions into breakpoint locations when breakpoints are set, so that a breakpoint-handling routine is executed when program execution reaches a breakpoint location. The modifications made by CID at a breakpoint location are not displayed by the DISPLAY command; that is, an attempt to display the contents of a breakpoint location causes the previous contents of the location to be displayed. Modification of the contents of the breakpoint location by your program destroys the breakpoint detection mechanism. However, since the location is still known to be a breakpoint by CID, the modifications are not displayed.

If a breakpoint location contains an RJ instruction, breakpoint detection occurs after any prior instructions in the word have been executed except for the first time detection (but not necessarily action) occurs.

Whether or not the location contains an RJ instruction, all parts of the breakpoint word that could be used to supply information to a called routine are valid.

You should be careful when you set a breakpoint at an entry point that is the target of an RJ instruction, because the breakpoint is destroyed when the exit jump is stored by the RJ instruc-

tion. Such a breakpoint is useful only if the subroutine has already been entered and detection of subroutine exit is desired. You should clear the breakpoint before returning from the subroutine. (Another way to detect a subroutine exit is to set an RJ trap described later in this section. The RJ trap can be set while either inside or outside the subroutine.) A warning message is issued when a breakpoint is set at a location that is an entry point but not explicitly designated as such in the location parameter of the SET,BREAKPOINT command.

Displaying the debug variable #P (see section 4) on an INSTRUCTION, RJ, XJ, JUMP, STORE, or FETCH trap at a breakpoint location appears no differently than when no breakpoint is set there.

The return jump to the breakpoint entry is not trapped by any RJ trap or INSTRUCTION trap. CID makes breakpoints invisible in this respect as well.

TRAPS

A trap automatically suspends program execution when a special condition occurs. You can then observe and change program values at the point where program execution was suspended.

The setting of a trap allows for a specific condition to be monitored over a specified program region without disturbing the logic of the program being executed. With one exception, the conditions monitored are under the control of the program being debugged. The single exception is that the INTERRUPT trap allows a program to be stopped any time you issue an interrupt signal from the terminal.

The SET,TRAP command sets a trap, the CLEAR,TRAP command removes one or more traps, the LIST,TRAP command lists one or more traps at the terminal, and the SAVE,TRAP command saves one or more traps on a local file. These commands are described in section 5.

The SET,TRAP command (described in section 5) has these parameters: type, scope, and report-level. The type parameter specifies what condition causes the trap to occur, the scope parameter specifies what locations are to be monitored for the trap condition, and the report-level parameter specifies what kind of address CID displays when the trap occurs.

When a trap is set, the trap is assigned a number in the range 1 through 16. This trap number, referred to in the form #n, provides a convenient way of referring to traps in CID commands. The trap number is also used in the trap report message that CID issues when a trap occurs. A trap remains established for the remainder of the debug session, unless the trap is redefined by another SET,TRAP command or cleared by a CLEAR,TRAP command (see section 5).

TRAP TYPES

The following paragraphs describe the types of traps available with CID.

ABORT Trap

The ABORT trap occurs after abnormal program termination (for example, termination as a result of an execution-time error). The scope of an ABORT trap is an execution address range. A default ABORT trap with unrestricted scope exists before you set an ABORT trap. If you set an ABORT trap, the default ABORT trap remains in effect at all locations not in the scope of the ABORT trap that you set.

If your program is executing when a time limit occurs, control goes to CID and an ABORT trap is issued with a message:

```
*CP TIME LIMIT
```

If CID is executing when a time limit occurs, the message issued is:

```
*TIME LIMIT
```

With the TIME LIMIT message, you are asked to enter T to continue or CR to stop. If CR is entered, CID gains control, because of the time limit, and terminates the session. If T, or T,n is entered, NOS extends the time; CID is unaffected and continues executing.

A reprieve mechanism is used to implement the ABORT trap. CID uses the system routine RPV to gain control on abnormal termination.

You can designate reprieve routines to receive control on abnormal termination in FORTRAN, BASIC, and COMPASS programs. FORTRAN programs can designate reprieve routines by calling the subroutine RECOVR at the start of program execution with appropriate parameters (see the FORTRAN reference manual). BASIC programs can designate reprieve routines by executing the ON ERROR statement (see the BASIC reference manual). COMPASS programs can designate reprieve routines by calling the RECOVR macro. COBOL programs do not generally designate reprieve routines; when a phrase such as the ON OVERFLOW phrase is used in a COBOL program, the condition specified by the phrase does not cause an ABORT trap.

In programs that contain reprieve code, CID receives control on an ABORT trap before the reprieve code is executed. If no reprieve code is present, the EXECUTE (or GO) command with no location specified is not allowed after an ABORT trap. If there is reprieve code for abnormal termination the EXECUTE (or GO) command begins execution of the reprieve code. Upon the completion of the reprieve code, CID again receives control and reports that the reprieve code has been completed. On an ABORT trap, the debug variables #ERRCODE and #CPUERR contain values indicative of the reason for the abort (see section 4).

END Trap

The END trap occurs after normal program termination (that is, when program execution ends, not as the result of an error). The scope of an END trap is an execution address range. A default END trap with unrestricted scope exists before you set an END trap. If you set an END trap, the default END trap remains in effect at all locations not in the scope of the END trap that you set.

A reprieve mechanism is used to implement the END trap. CID uses the system routine RPV to gain control on normal termination.

You can designate reprieve routines to receive control on normal termination in FORTRAN programs by calling the subroutine RECOVR at the start of program execution with appropriate parameters (see the FORTRAN reference manual). BASIC and COBOL programs do not generally have reprieve code for normal termination.

In programs that contain reprieve code, CID receives control on an END trap prior to the execution of any reprieve code. If no reprieve code is present, the EXECUTE (or GO) command with no location specified is not allowed after an END trap. If there is reprieve code for normal termination, the EXECUTE command begins execution of the reprieve code. Upon the completion of the reprieve code, CID again receives control and reports that the reprieve code has been completed.

FETCH Trap

The FETCH trap occurs after data is fetched from a location within the trap scope. The scope of a FETCH trap is an address range. Note that the smallest possible scope for a FETCH trap is one memory word; if a data item or variable is specified as the scope of a FETCH trap, the FETCH trap can occur as the result of a fetch of a data item or variable that shares a word with the data item or variable specified as the scope. This situation occurs often with COBOL data items. The FETCH trap turns on interpret mode (described later in this section).

INSTRUCTION Trap

The INSTRUCTION trap occurs before each machine instruction in the trap scope is executed. The trap scope is an execution address range. The INSTRUCTION trap turns on interpret mode (described later in this section).

INTERRUPT Trap

The INTERRUPT trap occurs after you issue a terminal interrupt (see glossary, appendix C) during program execution. The scope of an INTERRUPT trap is an execution address range. A default INTERRUPT trap with unrestricted scope exists before you set an an INTERRUPT trap. If you set an INTERRUPT trap, the default INTERRUPT trap remains in effect at all locations not in the scope of the user-set INTERRUPT trap.

INTERRUPT trap action differs depending on whether one of the loaded programs is a BASIC, COBOL, or FORTRAN program compiled for use with CID. If none of the loaded programs is compiled for use with CID, execution is suspended after the currently executing machine instruction is complete.

If at least one of the loaded programs is a BASIC, COBOL, or FORTRAN program compiled for use with CID, the following considerations apply when you enter a single interrupt during program execution.

- If the executing program is not accepting terminal input, execution is suspended at the beginning of the next noncontinued, executable line in a program compiled for use with CID.
- If the executing program is accepting terminal input, the INTERRUPT trap occurs immediately, before the input line is read. If execution is resumed at the point of the interrupt, the entire input line must be reentered.
- If interrupt reprieve code has been specified (through execution of a BASIC ON ATTENTION statement or a call to the RECOVR routine), the reprieve code is executed when a GO or EXECUTE command with no location is entered.
- If the executing program is sending output to the terminal, the output in the output buffer is lost. More output might remain to be sent to the buffer; execution is not suspended until the statement causing the output has finished execution.

If you enter a second interrupt when one of the loaded programs is compiled for use with CID, execution is suspended immediately after the current machine instruction. Entering a second interrupt is useful in the following situations:

- The executing program is not compiled for use with CID and is executing an infinite loop. Control in this case would never return to a program that can be interrupted with a single interrupt.
- The executing program is sending a large quantity of output to the terminal, and you wish to ignore the remaining output. The second interrupt causes suspension before the statement causing the output is complete. If you specify the next line when you resume execution, you do not receive the remaining output.

You can also interrupt execution of command sequences, but this type of interrupt is not a trap. (See section 7.)

JUMP Trap

The JUMP trap suspends program execution before a machine-level jump instruction is executed if the jump is to take place. The scope of a JUMP trap is an execution address range. The JUMP trap turns on interpret mode (described later in this section).

LINE Trap

The LINE trap suspends program execution before each BASIC, COBOL, or FORTRAN source line is executed. The LINE trap type is available only for BASIC, COBOL, and FORTRAN programs compiled for use with CID. The scope of a LINE trap is an execution address range. The LINE trap occurs only at the beginning of lines that are not continued from previous lines. In FORTRAN and BASIC programs, the LINE trap only occurs at the beginning of lines containing executable statements. In COBOL programs, the LINE trap also occurs at procedure-name lines.

OVERLAY Trap

The OVERLAY trap suspends program execution after a specified FORTRAN overlay is loaded. The scope of an OVERLAY trap is a pair of octal overlay level designators of the form (p,s) or an asterisk to indicate all overlays.

When executing under CID control, the FORTRAN library routine OVERLAY gives control to CID when called. CID then checks if the overlay just loaded is one that should be trapped.

PROCEDURE Trap

The PROCEDURE trap suspends program execution when execution reaches the beginning of a COBOL paragraph or section in the procedure division. The PROCEDURE trap type is available only for COBOL programs compiled for use with CID. The scope of a PROCEDURE trap is an execution address range.

When a COBOL program is compiled for use with CID, executable code identifying the beginning of each paragraph and section in the procedure division is produced. The PROCEDURE trap occurs after this code has been executed.

RJ Trap

The RJ trap suspends program execution before a machine-level return jump (RJ) instruction is executed or an EQ 0,0 instruction in parcel 0 is executed. (The RJ instruction is used to call a subroutine; the EQ 0,0 is usually used to return from a subroutine.) The scope of an RJ trap is an execution address range. The RJ trap turns on interpret mode (described later in this section).

STORE Trap

The STORE trap suspends program execution after data is stored into a location within the trap scope. The scope of a STORE trap is an address range. Note that the smallest possible scope for a STORE trap is one memory word; if a data item or variable is specified as the scope of a STORE trap, the STORE trap can occur as the result of a store to a data item or variable that shares a word with the data item or variable specified as the scope. This situation occurs often with COBOL data items. The STORE trap turns on interpret mode (described later in this section).

XJ Trap

The XJ trap suspends program execution before a machine-level exchange jump (XJ) instruction is executed. (The RJ instruction is used to call a subroutine; the EQ 0,0 is usually used to return from a subroutine.) The scope of an XJ trap is an execution address range. The XJ trap turns on interpret mode (described later in this section).

DEFAULT TRAPS

For the trap types END, ABORT, and INTERRUPT, default traps always exist. The scope of a default trap is the balance of the address range not contained in the scope of any traps of the same type set by you. This means that in the absence of any END, ABORT, or INTERRUPT trap set by you, the corresponding default trap has unrestricted scope. If you set a trap of one of these types, and supply an asterisk (*) as the scope parameter, then the corresponding default trap no longer has any scope at all; that is, the default trap never occurs.

The default traps have report level L if at least one BASIC, COBOL, or FORTRAN program compiled for use with CID is loaded; otherwise, the report level is P.

TRAP ACTION

When the condition specified in an established trap is encountered in an executing program, program execution is suspended, and the suspended program is designated the home program. If a trap body (described later in this section) is not associated with the trap, CID issues a trap report message and a question mark prompt to indicate that you can enter CID commands. The trap report message is shown in figure 3-1. If a trap body is associated with the trap, the CID commands in the trap body are executed automatically, and you do not gain control, unless a PAUSE command is executed in the trap body. If you do not gain control, no trap message is issued.

```
*T #n, type trap-message xx location
```

n	Trap number.
type	Trap type in the form as specified in the SET,TRAP command (either keyword or abbreviation).
trap-message	One of the following:

Trap Type	Trap Message
STORE or FETCH	Memory location
ABORT	Reason for abort
OVERLAY	Number of overlay loaded
other	Null

xx	Either AT or IN (see text).
location	Identifies where the trap occurred; that is, where execution stopped. For message level P, location is reported as: [(p,s)][progname]_nnnnnB. The progname is omitted when the home program name remains unchanged since the last time CID had control. The overlay numbers (p,s) are supplied only if progname is supplied and the overlay is different from when CID last had control. The underscore always appears in this message even when the program name is absent. This specifies that the location is relative to the home program rather than to RA.

Figure 3-1. Trap Report Message

Details about trap action vary depending on the scope and location parameters entered in the SET,TRAP command when the trap is set. These details involve the home program designation and the trap message format.

Trap Action for BASIC Programs

This subsection describes details about trap action when the scope parameter in the SET,TRAP command specifies locations in a BASIC program compiled for use with CID. The trap condition causing execution of a BASIC program to be suspended can occur in the BASIC program itself or in a system routine called by the program (for example, a routine called as the result of a PRINT statement in the BASIC program). Designation of the home program and the trap message format depend on the report level parameter specified in the SET,TRAP command when the trap is set.

The report level parameter in the SET,TRAP command can specify one of three values:

- L (line number)
- S (statement number)
- P (program offset)

For BASIC programs compiled for use with CID, the default report level is L.

If the L report level is specified in the SET,TRAP command, the trap location is reported in the following form providing the home program has not changed:

    L.nnn

If the home program has changed as a result of the trap, the trap location is reported in the following form:

    P.prog_L.nnn.

The keyword AT in the trap message indicates that the next instruction to be executed is at the beginning of the BASIC source line named in the trap message. The keyword IN indicates that the next instruction is inside the BASIC source line. The BASIC program is designated the home program.

If the S report level is specified in the SET,TRAP command, the trap location is reported as follows:

    L.nnn (S.nnn)

or P.prog_L.nnn (S.nnn), if the home program has changed as a result of the trap. The line number is reported in both L.nnn and S.nnn format.

If the P report level is specified in the SET,TRAP command, the location where the trap condition occurred is reported as a module relative address (P.prog_nn), and the program in which the trap condition occurred is made the home program. The keyword AT in the trap message indicates that the next instruction to be executed is at the beginning of a word. The keyword IN indicates that the next instruction to be executed is not at the beginning of the word.

If the L or S report level is in effect, the BASIC program is designated the home program, even though the debug variable #P (described in section 4) is in some other program. The trap report indicates that the trap occurred in a BASIC statement.

Trap Action for COBOL Programs

This subsection describes details about trap action when the scope parameter in the SET,TRAP command specifies locations in a COBOL program compiled for use with CID. The trap condition causing execution of a COBOL program to be suspended can occur in the COBOL program itself or in a system routine called by the program (for example, a routine called as the result of a READ statement in the COBOL program). Designation of the home program and the trap message format depend on the report level parameter specified in the SET,TRAP command when the trap is set.

The report level parameter of the SET,TRAP command can have one of three values:

- L (line number)
- PR (procedure name)
- P (program offset)

For COBOL programs compiled for use with CID, the default report level is L.

If the L report level is specified in the SET,TRAP command, the trap location is reported in the following form providing the home program has not changed:

    L.nnn

If the home program has changed as a result of the trap, the trap location is reported as follows:

    P.prog_L.nnn

The keyword AT in the trap message indicates that the next instruction to be executed is at the beginning of the COBOL source line named in the trap message. The keyword IN indicates that the next instruction is inside the line or lines associated with the line number. (The lines associated with a line number begin with the numbered line and end with the first line where a COBOL statement is not continued.)

If the PR report level is specified in the SET,TRAP command, the trap location is reported in one of the following forms providing the home program has not changed:

- L.nnn (PR.procedure-name)
- L.nnn (m BEFORE PR.procedure-name)
- L.nnn (m AFTER PR.procedure-name)

The m is the number of executable and procedure-name lines before or after the beginning of the named paragraph or section.

If the home program has changed as a result of the trap, the trap location is reported in one of the following forms:

- P.prog_L.nnn (PR.procedure-name)
- P.prog_L.nnn (m BEFORE PR.procedure-name)
- P.prog_L.nnn (m AFTER PR.procedure-name)

If the P report level is specified in the SET,TRAP command, the location where the trap condition occurred is reported as a module relative address (P.prog nn), and the program in which the trap condition occurred is made the home program.

If the L or PR report level is in effect, the COBOL program is made the home program, even if the debug variable #P (described in section 4) contains an address in some other program. The trap report indicates that the trap occurred in the COBOL program.

Trap Action for FORTRAN Programs

This subsection describes details about trap action when the scope parameter in the SET,TRAP command specifies locations in a FORTRAN program compiled for use with CID. The trap condition causing execution of a FORTRAN program to be suspended can occur in the FORTRAN program itself or in a system routine called by the program (for example, a routine called as a result of a READ statement in the FORTRAN program). Designation of the home program and the trap message format depends on the report level parameter specified in the SET,TRAP command when the trap is set.

The report level parameter in the SET,TRAP command can specify one of the following values:

- L (line number)
- S (statement number)
- P (program offset)

For FORTRAN programs compiled for use with CID, the default report level is L.

If the L report level is specified in the SET,TRAP command, the trap location is reported in the following form providing the home program has not changed:

L.nnn

If the home program has changed as a result of the trap, the trap location is reported as follows:

P.prog_L.nnn.

The keyword AT in the trap message indicates that the next instruction to be executed is at the beginning of the FORTRAN source line named in the trap message. The keyword IN indicates that the next instruction is inside the line or lines associated with the line number. (The lines associated with a line number begin with the numbered line and end with the first line where a FORTRAN statement is not continued.) The FORTRAN program is made the home program.

If the S report level is specified in the SET,TRAP command, the trap location is reported in one of the following forms providing the home program has not changed:

- L.nnn (S.kk)
- L.nnn (m BEFORE S.kk)
- L.nnn (m AFTER S.kk)

The m is the number of executable lines before or after the referenced statement label. The nearest labeled statement is referenced. If the home program has changed as a result of the trap, the program name is also reported in the trap message.

If the P report level is specified in the SET,TRAP command, the location where the trap condition occurred is reported as a module relative address (P.prog_nn), and that program is made the home program.

If the L or S report levels are in effect, the FORTRAN program is designated the home program, even if the debug variable #P (described in section 4) is in some other program. The trap report indicates that the trap occurred in a FORTRAN statement.

Trap Action for Other Programs

This subsection describes details about trap action for programs other than BASIC, COBOL, and FORTRAN programs compiled for use with CID. The report level for traps in this kind of program must be P (P is selected by default when no report level is specified in the SET,TRAP command).

In the trap report message, the location where the trap condition occurred is reported as a module relative address (P.prog_nn), and the program in which the trap occurred is made the home program. The keyword AT in the trap message indicates that the next instruction to be executed is at the beginning of a word. The keyword IN indicates that the next instruction to be executed is not at the beginning of the word.

INTERPRET MODE

Interpret mode of program execution is the mechanism used to detect the occurrence of an INSTRUCTION, RJ, XJ, JUMP, FETCH, or STORE trap. In interpret mode, each machine instruction is simulated by an interpreter routine. The contents of all registers are also simulated by the interpreter. Execution in interpret mode takes as much as several hundred times longer than normal execution. Interpret mode is turned on when the first trap of one of these types is established and remains on until all such trap types have been removed with a CLEAR,TRAP command or until interpret mode has been explicitly turned off by a SET,INTERPRET,OFF or CLEAR,INTERPRET command. These commands are described in section 5. SET,INTERPRET,OFF deactivates traps but does not remove them.

COMMAND SEQUENCES

A command sequence is a sequence of commands that are grouped and executed together. Command sequences can be classified as breakpoint or trap bodies, groups, or line sequences.

BREAKPOINT AND TRAP BODIES

Often, whenever a particular breakpoint or trap condition occurs, you wish to have the same set of CID commands executed. To avoid repeatedly entering the same commands each time the condition occurs, the SET,BREAKPOINT or SET,TRAP commands provide for declaring a body (or sequence) of CID commands to be automatically executed when the trap or breakpoint occurs.

In this case, you are not notified when the trap or breakpoint occurs; you do not gain control at any time during the program interruption unless a PAUSE command is encountered during execution of the CID command sequence. Program execution continues after the last command in the command sequence is executed or after a command that explicitly resumes program execution is executed.

GROUPS

A certain command sequence may be so useful that it is required as part of several command sequences in various breakpoint or trap declarations. You may also need to execute a group of commands several times when in interactive mode. A command sequence called a group provides this capability. The SET,GROUP command can be entered to create a group. The READ command executes the CID commands contained in a group (see section 5).

LINE SEQUENCES

A line sequence is a sequence of commands all on one line. The commands must be separated by semicolons.

Special responses are available if an error, warning, or interrupt occurs while processing a line sequence. These responses are described in section 7.

COLLECT MODE

Collect mode is a mode in which you create trap bodies, breakpoint bodies, and groups. You can activate collect mode when you enter the SET,BREAKPOINT, SET,TRAP, or SET,GROUP command. In the SET,BREAKPOINT and SET,TRAP commands, you activate collect mode by specifying a left bracket ([) in the command line (see section 5). In the SET,GROUP command, collect mode is activated whether or not the left bracket is specified. When collect mode is activated, CID issues the message IN COLLECT MODE.

In collect mode, issued commands are not processed but are syntax checked. All commands that are free of syntax errors are collected and stored on a special file, one command per line, even if originally input with more than one command per line. Once activated, collect mode remains in effect until a right bracket (]) is encountered at the first level of collect mode. When the right bracket is encountered, CID issues the message END COLLECT.

Increasing levels of collect mode are defined for each additional command that establishes a trap or breakpoint with a body or a group. In this case, collect mode remains in effect until enough right brackets are encountered to balance the left brackets. (The left brackets are implicit in a SET,GROUP command.)

CID SEQUENCE EXECUTION

Once a CID command sequence has begun executing, execution of the sequence continues automatically through the sequence until the end of the sequence is reached, or until a GO, EXECUTE, or QUIT command is encountered.

The command sequence can be suspended by a PAUSE, SUSPEND, or READ command (see section 5), or by a terminal interrupt (see section 7). Command sequences can be nested to 16 levels.

COMMAND SEQUENCE FILES

It is possible to prepare a sequence of CID commands on a file formatted exactly as you would enter them from the terminal. This could be done using some text file utility such as a text editor. Such a file, when attached to the job, is executed by a READ command. With a file prepared in this manner, any multiple command lines are maintained as such. Thus, LABEL commands can only appear as the first command on a line.

COMMAND SEQUENCE EXAMPLES

The example in figure 3-2 shows an RJ trap with a body sequence. In the example, subroutine calls are distinguished from subroutine exits by examining the operation code. The first two levels of subroutine calls are reported when they occur. The entry points of additional calls are stored in the debug user variables #V1 through #V9 (debug user variables are described in section 4). If calls are nested more than 11 deep, a message and pause results. When an exit is encountered, the call level is reduced by one.

The example in figure 3-3 is a defined group designed for use with the trap in figure 3-2. Whenever the CID command READ,TRACEIT is issued, a traceback of up to nine levels is displayed. This traceback scheme prevents ambiguity arising from multiple entry points in a module.

EDITING A COMMAND SEQUENCE

If you discover that a change is needed in a lengthy command sequence, you can reenter the entire definition, while still under CID control.

```
SET,TRAP,RJ,* [
SKIPIF,#OP,EQ,1; JUMP,SUBEXIT
ENTER,#V10+1,#V10
SKIPIF,#V10,GT,2 ; JUMP,DISP
LABEL,STORE
SKIPIF,#V10,LE,11; JUMP,NOROOM
SKIPIF,#V10,NE,3 ; JUMP,1
SKIPIF,#V10,NE,4 ; JUMP,2
SKIPIF,#V10,NE,5 ; JUMP,3
SKIPIF,#V10,NE,6 ; JUMP,4
SKIPIF,#V10,NE,7 ; JUMP,5
SKIPIF,#V10,NE,8 ; JUMP,6
SKIPIF,#V10,NE,9 ; JUMP,7
SKIPIF,#V10,NE,10; JUMP,8
SKIPIF,#V10,NE,11; JUMP,9
LABEL,1; ENTER,#EA,#V1; GO
LABEL,2; ENTER,#EA,#V2; GO
LABEL,3; ENTER,#EA,#V3; GO
LABEL,4; ENTER,#EA,#V4; GO
LABEL,5; ENTER,#EA,#V5; GO
LABEL,6; ENTER,#EA,#V6; GO
LABEL,7; ENTER,#EA,#V7; GO
LABEL,8; ENTER,#EA,#V8; GO
LABEL,9; ENTER,#EA,#V9; GO
LABEL,NOROOM
MESSAGE, "  TRACEBACK OVERFLOW  "
PAUSE
GO
LABEL,SUBEXIT
SKIPIF,#V10,LT,1
ENTER,#V10-1,#V10
GO
LABEL,DISP
MESSAGE, "S/R CALL AT "
DISPLAY,#EA,A
]
```

Figure 3-2. CID Sequence Example

```
SET,GROUP,TRACEIT
MESSAGE, "  STORED TRACEBACK BEGINS  "
SKIPIF,11,GT,#V10; DISPLAY,#V9,A
SKIPIF,10,GT,#V10; DISPLAY,#V8,A
SKIPIF, 9,GT,#V10; DISPLAY,#V7,A
SKIPIF, 8,GT,#V10; DISPLAY,#V6,A
SKIPIF, 7,GT,#V10; DISPLAY,#V5,A
SKIPIF, 6,GT,#V10; DISPLAY,#V4,A
SKIPIF, 5,GT,#V10; DISPLAY,#V3,A
SKIPIF, 4,GT,#V10; DISPLAY,#V2,A
SKIPIF, 3,GT,#V10; DISPLAY,#V1,A
MESSAGE, "  STORED TRACEBACK ENDS  "
]
```

Figure 3-3. Defined Group

However, a simpler way would be to use the system editor to make the required changes. This can be done if the following steps are performed:

1. Save the definition on a file using the SAVE command.
2. Issue a SUSPEND command.
3. Use the editor to make the desired changes to the definition on the saved file.
4. Issue a DEBUG(RESUME) system command. This will reactivate CID at the point of suspension.
5. Issue an appropriate CLEAR command to remove the old definition.
6. Issue a READ on the saved file containing the edited definition. This will reconstitute the definition establishing it in its revised form.

SYNTAX OF LANGUAGE-INDEPENDENT COMMANDS 4

This section describes the syntax of language-independent CID commands. Descriptions of the language-independent commands themselves are contained in section 5.

FORMAT OF LANGUAGE-INDEPENDENT COMMANDS

CID commands begin with the command name (keyword). Many language-independent commands have a short form that can be entered in place of the command name. Normally, each line entered from the terminal is considered a complete command. More than one command can be entered on a line if a semicolon separates the commands. Language-independent and language-dependent commands can be entered on the same line.

Commands cannot be continued across lines. The maximum length of a command line is 150 6-bit characters. Characters beyond 150 are ignored.

CID can be used from a NOS ASCII mode terminal. The command line is limited to a maximum of 75 12-bit escape code ASCII characters. CID can be used to debug ASCII mode BASIC programs under NOS. NOS/BE, however, does not support input or output of lowercase ASCII characters with CID.

If an asterisk appears as the first nonblank character of a line, or as the first nonblank character after a semicolon, the remaining characters are considered to be a comment.

Parameters of language-independent commands must be separated from the command name and from each other by a comma, a space, or both. Excess spaces are ignored between parameters. (Examples in this manual show a comma as the delimiter between parameters, although a space might also appear to improve readability.)

The COBOL CID commands described in section 6 have the same names as many of the language-independent commands described in section 5. To avoid ambiguities when the home program is a COBOL program compiled for use with CID, you should enter language-independent commands in short form or with a comma following the name of the command. CID assumes any ambiguous command is a COBOL CID command when the home program is a COBOL program compiled for use with CID.

Parameters in language-independent commands are order dependent. Optional parameters at the end of a parameter list can be omitted entirely. However, when optional parameters within a list are omitted, the positions of the parameters must often be acknowledged by commas. The figures in the command descriptions show whether or not the commas are necessary.

ADDRESSES

In language-independent commands, addresses can be specified as absolute addresses, module relative addresses, entry point addresses, overlay addresses, or source language symbol addresses. A range of addresses can also be specified in most CID commands.

ABSOLUTE ADDRESSES

The following numeric literal notations are available for conveying addresses relative to the start of the field length:

- Central Memory Field Length:

Form	Base
n	Decimal
nD	Decimal
mB	Octal

- ECS/LCM Field Length:

Form	Base
X.n	Decimal
X.nD	Decimal
X.mB	Octal

where n is a decimal numeral of up to 17 digits and m is an octal numeral of up to 20 digits.

Examples of absolute addresses are:

29	CM address 29
29D	CM address 29
35B	CM address 29 (equals 35_8)
X.48	ECS/LCM address 48
X.48D	ECS/LCM address 48
X.60B	ECS/LCM address 48 (equals 60_8)

When used in a parameter requiring a value, these numeric literals supply the address as the value. When used as an address, only the lower 18 bits are used for central memory addresses, and the lower 24 bits for ECS/LCM addresses.

MODULE RELATIVE ADDRESSES

You can specify addresses relative to loader blocks or modules. Names of modules and entry points can be directly specified if the names contain only letters and digits and begin with a letter. Names not conforming to this rule can be expressed as

literals delimited by dollar signs ($). For example:

$AB.CD$ is the equivalent of AB.CD.

Specific address locations within a loader block or module are designated as follows:

Module Type	Designation
Program module	P.progname_n
Labeled common block (central memory)	C.blockname_n
Labeled common block (extended memory)	XC.blockname_n
Blank common block	C._n

- The underscore (_) in CID indicates a relative address in a program module or common block. The symbol to the left of the underscore specifies a block or module name. The symbol to the right of the underscore specifies the relative address within the block or module.
- The n is an integer constant which is the address of the desired location relative to the start of the block or module. (That is, C._0 is word 0, the first word of the unlabeled common block.)
- For a block or module L words long, the maximum allowed value of n is L-1. The error RELATIVE ADDRESS OUTSIDE BLOCK results if n is larger than L-1. (L-1 references the last word of the block or module.)
- Other notations for n are as follows:

  nD (decimal notation)

  nB (octal notation)
- When overlays are used, modules in separate overlays can have the same name. A specific module can be specified by prefixing the module designation by an overlay designation, as described later in this section.

  In the absence of an overlay qualifier, the selection is made from the overlays currently loaded.

Examples of module relative addresses are:

- C.BLKA_9

  The 10th word of common block BLKA
- P.PROGC_22B

  The 19th word of program PROGC (22_8 is 18)

As a notational shorthand convenience, P.program name can be omitted in a program module relative address when the program intended is that designated as the home program.

Whenever CID obtains control from your program, the program module most recently in execution is automatically made the home program. The SET,HOME command can be entered to designate some other program module as the home program.

Examples of references to addresses in the home program are:

_0 The first location in the home program

_72B Relative location 72_8 words after the first word in the home program

ENTRY POINT ADDRESSES

Entry point locations are designated as follows:

E.entryname

where entryname is the name of the entry point.

While duplicate entry point names are allowed in a load, the Loader always uses the first occurrence of an entry point to link references. CID follows the same convention; thus, duplicates will not be accessible.

Examples of entry point addresses are:

- E.START

  Location of entry point START
- E.FRED

  Location of entry point FRED

OVERLAY ADDRESSES

Overlays are designated on CID input and output using the same scheme as in FORTRAN and COMPASS directives, namely:

Overlay Type	Designated As
main	(0,0)
primary	(p,0)
secondary	(p,s)

where p and s are positive integers.

Program names can be qualified with an overlay prefix. For example:

(p,s)P.PROGA

This indicates the program named PROGA in overlay (p,s). Output reports qualify locations by designating the overlay as follows:

P.PROGA_L.4 (OF (2,5))

References to an overlay that is not currently loaded can be made only in the following commands:

- CLEAR,BREAKPOINT
- CLEAR,TRAP
- LIST,BREAKPOINT
- LIST,MAP
- LIST,TRAP
- SAVE,BREAKPOINT
- SAVE,TRAP

- SET,BREAKPOINT
- SET,HOME
- SET,TRAP

SOURCE LANGUAGE SYMBOL ADDRESSES

Symbols contained in a source program can be entered in language-independent commands when the program is a BASIC, COBOL, or FORTRAN program compiled for use with CID.

BASIC Symbols

You can enter statement numbers and variable names in language-independent commands to reference locations within BASIC programs compiled for use with CID.

A statement reference refers to the beginning of a BASIC source statement and has the following format:

    L.n or S.n

The n is the line number of the BASIC line. L and S stand for line and statement, respectively. L.n and S.n can be used interchangeably when referring to BASIC statements because line numbers and statement labels are synonymous in BASIC. The value of a statement reference is the address of the beginning of the statement.

A variable name reference consists of just the name as it appears in the BASIC program. The value of a variable name reference is the address of the named variable. When an array name is the same as a simple variable name, the simple variable is used unless a specific array element is referenced. Subscripts in language-independent commands must be constants.

Because BASIC stores array elements in row order, all the elements of one row are contiguous in memory. The address range B(1,2)...B(1,5), therefore, contains elements B(1,2), B(1,3), B(1,4), and B(1,5). The range B(1,1)...B(3,1) contains all the elements of the first two rows plus element B(3,1).

Variables that are function formal parameters are only known when the program is executing the function. A trap or breakpoint must occur inside a function before its formal parameters can be referenced in CID commands.

String variables are referenced in the same way as numeric variables, by using the variable name. Special care must be taken when using string variables in language-independent commands such as DISPLAY and ENTER. BASIC stores its string data in a dynamic string memory area. String variables contain pointers to the dynamic string area rather than string data. DISPLAY, ENTER, and MOVE commands, therefore, affect the string pointer word rather than the string. These commands should not be used to manipulate BASIC string variables. Instead, the BASIC CID PRINT and LET commands described in section 6 should be used. Substring notation is not available in language-independent commands.

COBOL Symbols

You can enter line numbers, procedure names, and identifiers to reference locations within COBOL programs compiled for use with CID.

A line reference refers to the beginning of a COBOL source line and has the following format:

    L.n

where n is the sequence number of the COBOL source line being referenced. The sequence number is assigned by you if the PSQ option is used in the COBOL5 control statement or by the compiler if the PSQ option is not used. A line reference cannot reference a line that contains a statement continued from a previous line. The value of a line reference is the address of the beginning of the line.

A procedure reference is a reference to a COBOL paragraph or section in the Procedure Division. A procedure reference has the following format:

    PR.procname

where procname is a COBOL procedure name (that is, the name of a paragraph or section in the procedure division) The forms procname can take are:

- Paragraph name
- Section name
- Paragraph name OF section name

The value of a procedure reference is the address of the beginning of the procedure. CID does not check for duplicate paragraph names in different sections when you specify an unqualified paragraph name.

An identifier reference is identical to an identifier for a data item in the COBOL program. Table items are specified by the table name followed by numeric constant subscripts. Reference modification cannot be used in language-independent commands. The value of an identifier reference is the address of the first word containing the data item. Qualification is allowed in identifier references.

FORTRAN Symbols

You can enter line numbers, statement labels, and variable names in language-independent commands to reference locations within FORTRAN programs compiled for use with CID.

A line number reference refers to the beginning of a FORTRAN source line and has the following format:

    L.n

The n is the compiler-assigned line number in the program unit (or the sequence number supplied by the program if the SEQ parameter was specified when the compiler was called). Line numbers can be determined from the compiler output listing. The value of a line number reference is the address of

the beginning of the FORTRAN line. Line number references can only reference lines that begin with executable statements that are not continued from a previous line.

A statement label reference has the following format:

    S.n

The n is a FORTRAN statement label. Only labels on executable statements are valid for referencing in CID commands. The value of a statement label reference is the address of the beginning of the statement.

A variable name reference consists of the variable name as it appears in the FORTRAN program. The value of a variable reference is the address of the named variable. Arrays can be referenced either by the name of the array (to indicate the whole array) or by specific array elements, depending on the context. Subscripts can only be constants in language-independent commands.

Because FORTRAN stores array elements in column order (the leftmost subscript varying fastest), the address range A(1,5)...A(5,5) includes elements A(1,5), A(2,5), A(3,5), A(4,5), and A(5,5). On the other hand, the address range A(1,1)...A(1,5) includes all elements in rows 1 through 4 plus element A(1,5) of row 5.

ADDRESS RANGE SPECIFICATION

An address range is a set of consecutive address locations used to designate a specific area in a program.

An address range can be specified in one of two ways:

- Module reference
- Ellipsis notation

MODULE OR BLOCK REFERENCING

The various types of modules or blocks that can be referenced in their entirety (that is, the entire address range where they are loaded) are as follows:

Module or Block Type	Designation
Program module	P.progname
Labeled common block (central memory)	C.blockname
Labeled common block (extended memory)	XC.blockname
Unlabeled common block	C.

Examples of module referencing are:

- C.BLKA

  The entire labeled common block is BLKA.

- XC.BLKX

  The entire common block is BLKX and is stored in extended memory.

- P.PROGB

  The entire program module is PROGB.

ELLIPSIS

The most general way to specify an address range is by using the following notation called the ellipsis notation:

    address expression...address expression

The ellipsis notation can contain two or more periods. In this manual, three periods are used. The ellipsis notation must obey the following rules:

- No spaces are allowed before, between, or following the periods.
- The first address must not be greater than the second one.
- The range must not span an overlay boundary.
- Both expressions must be for the same memory type (central or extended memory).

If the indicated range goes beyond the field length or extends into locations not allowed by CID, a warning message is issued. If an affirmative acknowledgment is issued in response, truncation of the range to accessible locations occurs.

An example of the ellipsis notation is:

    C.BLKA_0...C.BLKB_0-1

This example specifies the address range beginning with the first location in common block BLKA and ending with the location preceding the first location in common block BLKB.

If no other modules are loaded between BLKA and BLKB, the preceding address range specification can be written as:

    C.BLKA

VALUES

Values in language-independent commands can be specified as numeric constant values, address values, or debug variables.

NUMERIC CONSTANT VALUES

Numeric constants can be entered as values in language-independent commands. Numeric constants can be of types decimal integer or octal integer.

Decimal Integer Constants

A 60-bit integer constant (figure 4-1) consists of a string of digits optionally preceded by a plus sign (+) or a minus sign (-).

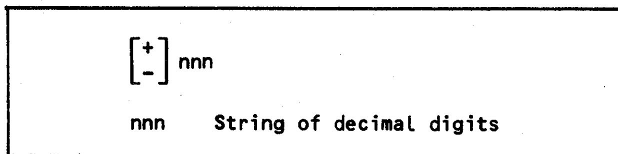


Figure 4-1. Decimal Integer Constant

Octal Integer Constants

A 60-bit octal constant (figure 4-2) is a string of 1 to 20 octal digits (0 through 7) followed by the letter B.

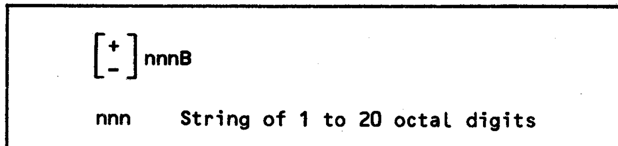


Figure 4-2. Octal Integer Constant

ADDRESS VALUES

Addresses can be used as values in language-independent commands. When an address is used as a value, the value consists of the address itself and not the contents of the address.

DEBUG VARIABLES

There are several available variables stored within CID itself. They all have identifiers beginning with #. Debug variables are classified in five categories:

- Debug user variables
- Debug state variables
- Program state variables
- Interpret mode variables
- Register state variables

Debug User Variables

Ten debug user variables, designated #V1, #V2,..., #V10, are available. They can be given values by the ENTER command (described in section 5) and used in expressions. Initially their values are zero.

Debug State Variables

Debug state variables provide information about the debug session. They are updated by CID and you cannot directly change them. However, some CID commands change the values of these variables. Debug state variables with numeric values can be used in expressions. The debug state variables are shown in table 4-1.

TABLE 4-1. DEBUG STATE VARIABLES

Variable	Can Use In Expressions	Description
#BP	Yes	Number of break-points currently defined
#GP	Yes	Number of groups currently defined
#HOME	No	Home program name
#TP	Yes	Number of traps currently defined

Program State Variables

Program state variables provide information about the state of the program. Each time program execution is suspended, the program state variables are updated by CID to reflect the current status of the program. You cannot change the values of the program state variables. The program state variables are shown in table 4-2. #LINE, #PROC, and #P are undefined before program execution is initiated at the beginning of a debug session. The variables #ERRCODE and #CPUERR are described in more detail in figure 4-3.

TABLE 4-2. PROGRAM STATE VARIABLES

Variable	Can Use In Expressions	Description
#CPUERR	Yes	Mode error code
#ERRCODE	Yes	Reprieve or execution-time generated error code
#FE	Yes	Extended memory field length
#FL	Yes	Central memory field length
#LINE	No	Current BASIC, COBOL, or FORTRAN line number
#P	Yes	Program address register
#PROC	No	Current COBOL pro-cedure name in the form of a para-graph name not qualified by a section name.

#ERRCODE System error code issued on an abort

This is meaningful only in an ABORT trap. This value can be displayed or used as a source of data, but it cannot be changed. When displayed with the DISPLAY command, the default is integer format, but other format options can be specified.

Value	Meaning
0	Normal termination
1	Time limit
2	CPU error exit
3	PP abort
4	CPU abort
5	PP call error
6	Operator drop
7	Operator kill
8	Operator rerun
9	Control statement error
10	ECS parity error
13	Auto-recall error
14	Job hung in auto-recall
15	Mass storage limit
16	PP program not in library
17	I/O limits
31	Invalid store, fetch, dump
32	Terminal interrupt
100 thru 299	Basic execution-time error
300	COBOL execution-time error

#CPUERR Hardware-detected mode error

This is meaningful only in an ABORT trap. This value can be displayed or used as a source of data, but cannot be changed. When displayed with the DISPLAY command, octal display mode is enforced regardless of the option specified.

Mode errors reported from job termination are as follows:

0	Program attempted to jump to location 0
1	Address referenced outside field length
2	Infinite operand used
3	Address out of field length, or infinite operand
4	Floating-point operand has undefined value
5	Address is out of field length or floating-point operand has undefined value
6	Infinite operand used or floating-point operand has undefined value
7	Infinite operand used, address referenced outside field length, or floating-point operand has undefined value
10	Program attempted to use an exchange jump instruction that does not exist on the hardware

Figure 4-3. Abort Information Variables

Interpret Mode Variables

Interpret mode variables provide more detailed program or machine status. They are updated by CID only when your program is executing in interpret mode. Thus, their values are meaningful only in this case. Interpret mode variables are listed and described in figure 4-4. The variables #EW, #PC, and #EA are particularly useful when STORE and FETCH traps (described in section 3) are used.

Register State Variables

Register state variables are updated by CID and reflect the current contents of the hardware registers at the time of suspension of program execution. These variables are listed and described in table 4-3.

EXPRESSIONS

Expressions are used in CID command parameters as an addresses or as values. For example, the syntax of the ENTER command, as described in section 5, is as follows:

```
ENTER,value,address
```

The first parameter is to be used as a value; the second as an address. Both parameters are specified with expressions.

When an expression is used as a value, all 60 bits of the expression result are used, and no checking is performed to determine if the value is a valid address.

When an expression is used as a central memory address, only the lower 18 bits of the expression

#EA	Effective address of current instruction. For a CM fetch, #EA is the address from which the word is fetched. For a CM store, #EA is the address where the word is stored. For a transfer of control instruction, #EA is the address where control is transferred. For other instructions, #EA is undefined. The value of #EA can be displayed or used as a source of data. When #EA is displayed with the DISPLAY command, the default display mode is type address. If an instruction such as SA6 K is trapped, then #EA has the value K.
#EW	Effective word. For a CM fetch or store, #EW is the value fetched or stored. Otherwise, #EW is undefined. This value can be displayed or used as a source of data. When displayed with the DISPLAY command, the default display mode is octal. If an instruction such as SA6 K is trapped, then #EW has the same value as #X6.
#I	i field of current instruction. This value can be displayed or used as a source of data, but it cannot be changed. When displayed with the DISPLAY command, the default display mode is octal.
#INS	Current instruction as a number. This value can only be displayed. When displayed with the DISPLAY command, the default display mode is octal.
#INSL	Current instruction length (15, 30, or 60). This value can be displayed or used as a source of data, but it cannot be changed. When displayed with the DISPLAY command, the default display mode is decimal.
#J	j field of current instruction. This value can be displayed or used as a source of data, but it cannot be changed. When displayed with the DISPLAY command, the default display mode is octal.
#K	k or K field of current instruction. This value can be displayed or used as a source of data, but it cannot be changed. When displayed with the DISPLAY command, the default display mode is octal.
#OP	Operation code of current instruction. This item is considered to be just two octal digits; therefore, in certain cases the i field of the instruction must be examined in order to determine the exact instruction. For example, RJ and ECS/LCM instructions all have #OP=01. This value can be displayed or used as a source of data, but it cannot be changed. When displayed with the DISPLAY command, the default display mode is octal.
#PA	Previous address. For a CM fetch or store, #PA is the previous address stored in the A register used to do the store or fetch. Otherwise, #PA is undefined. This value can be displayed or used as a source of data, but it cannot be changed. When displayed with the DISPLAY command, the default display mode is type address.
#PARCEL	Instruction parcel counter. Parcels are numbered left to right from 0 to 3. This value can be displayed or used as a source of data, but it cannot be changed.
#PC	Previous contents. For a CM store, #PC is the previous contents prior to the store; for a CM fetch, #PC is the X register value prior to the fetch. Otherwise, #PC is undefined. This value can be displayed or used as a source of data, but it cannot be changed. When displayed with the DISPLAY command, the default display mode is octal.

Figure 4-4. Interpret Mode Variables

TABLE 4-3. REGISTER STATE VARIABLES

Variable	Description
#A	All A registers. This form is only valid for display purposes. When displayed with the DISPLAY command, the default display mode is octal; the contents of all A registers are displayed.
#Ai	Register Ai where $0 \leq i \leq 7$. The data in the specified A register is regarded as a signed 18-bit number. These values can be displayed, used as a source of data, or changed. When displayed with the DISPLAY command, the default display mode is octal.
#B	All B registers. This form is only valid for display purposes. When displayed with the DISPLAY command, the default display mode is octal.
#Bi	Register Bi where $0 \leq i \leq 7$. The data in the specified B register is regarded as a signed 18-bit number. These values can be displayed, used as a source of data, or changed. When displayed with the DISPLAY command, the default display mode is octal.
#REG	Means all registers. This form is only valid for display purposes. When displayed with the DISPLAY command, the default display mode is octal.
#X	All X registers. This form is only valid for display purposes. When displayed with the DISPLAY command, the default display mode is octal.
#Xi	Register Xi where $0 \leq i \leq 7$. The data in the specified X register is regarded as a signed 60-bit number. These values can be displayed, used as a source of data, or changed. When displayed with the DISPLAY command, the default display mode is octal.

result are used. When it is used as an extended memory address, the lower 24 bits are used. Furthermore, the value is checked when it is used to ensure it constitutes a valid address.

Simple expressions in language-independent commands consist solely of an address reference, a debug variable, or a numeric constant value. More complicated expressions can be formed by combining one or more simple expressions with one or more operators. There are three operators: + , - , and !.

Parentheses can be used to group terms into a subexpression. When encountered, the subexpression is evaluated before proceeding further with the evaluation indicated by any pending operators or remaining terms.

All CID expressions evaluate to a signed 60-bit integer. Expressions must not contain any spaces.

ADDITION AND SUBTRACTION OPERATORS

The operators + and - can appear in an expression, indicating that addition or subtraction of the values of the adjacent terms is to be performed.

Individual terms are evaluated and algebraically summed in order left to right. Sixty-bit integer arithmetic is used in performing the indicated addition or subtraction. That is, no intermediate or final result can exceed a magnitude of 576460752303423487. No checks are made for overflow.

If + or - precedes the first or only term in an expression, the expression is evaluated as if a term preceding the operator existed with a value of 0. Thus, - in this case acts as a negation operator and + acts as an identity operator (returning its argument as its result).

Examples using the addition and subtraction operators are:

- #FL-1

  This expression is the highest location in the field length.

- E.PRGENT+1

  This expression evaluates to the address of entry point PRGENT plus 1, which is one word beyond entry point PRGENT.

- C.BLKA_0-1

  This expression evaluates to the address preceding common block BLKA.

VALUE OPERATOR (!)

In expressions, the exclamation point (!) is a prefix operator for the expression it prefixes. The lower 18 bits of the value are used as an address. The value stored at that address is the result.

Examples of the value operator are as follows:

!2	Value at address 2
!#P	Value at the current location

There must be no spaces between the (!) character and the address.

The value operator has a higher precedence than + or - , so that in multiple term expressions it is performed before + and -. For example, !2+3 means the same as 3+!2. The resulting expression is a value 3 greater than the value stored in the word at location 2.

If you wish to express the value stored at a location designated by a compound expression, parentheses can be used. For example:

```
!(C.BLKA_0+#V1)
```

This expression signifies the value stored at the location C.BLKA_0+#V1.

Multiple ! operators can be used to achieve indirect addressing. For example, !!C.BLKA_0 represents the value of the word whose address is stored at location C.BLKA_0. !C.BLKA_0 represents the value of the word at location C.BLKA_0, and ! applied to that value returns the word whose address is in location C.BLKA_0.

It is not possible to indirectly access one extended memory word through another; all indirect addresses are assumed to be central memory addresses.

LANGUAGE-INDEPENDENT COMMANDS 5

This section contains descriptions of the language-independent CID commands. Language-independent commands are classified as follows:

- CLEAR commands
- LIST commands
- SAVE commands
- SET commands
- Other language-independent commands

The syntax of language-independent commands is described in section 4.

CLEAR COMMANDS

CLEAR commands allow you to remove objects created via the SET command, such as breakpoints, traps, and groups. The variants of a CLEAR command can be expressed in abbreviated forms (see table 5-1). The general form of a CLEAR command is:

```
CLEAR,object,parameters
```

where object is the object to be cleared, and parameters is the (optional) list of qualifiers associated with that particular object.

CLEAR,AUXILIARY COMMAND

The CLEAR,AUXILIARY command (figure 5-1) deactivates the auxiliary output file. If an auxiliary output file has been defined using the SET,AUXILIARY command (described in this section), that file is closed when the CLEAR,AUXILIARY command is executed.

Figure 5-1. CLEAR,AUXILIARY Command

CLEAR,BREAKPOINT COMMAND

The CLEAR,BREAKPOINT command (figure 5-2) removes specified breakpoints. You can remove all breakpoints or a selection of breakpoints.

An example of the CLEAR,BREAKPOINT command is:

```
CB,P.WHALERS_44,P.BAY,(1,5),P.WHARF_26
```

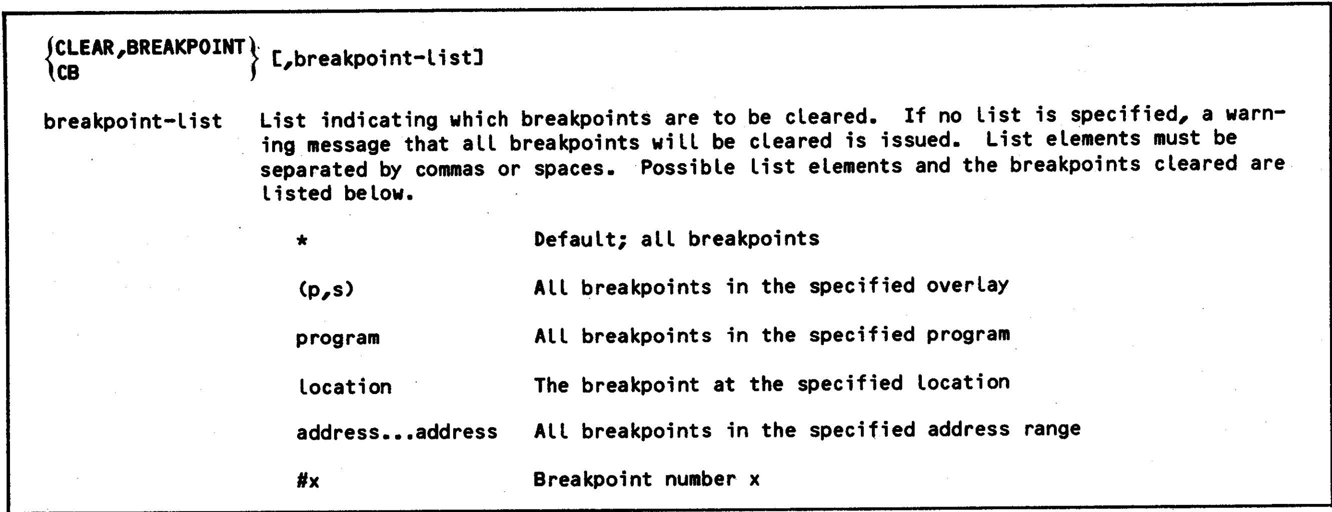
This command clears the single breakpoint from address 44 in program WHALERS, clears all breakpoints in program BAY, clears all breakpoints in the (1,5) overlay, and clears the single breakpoint from address 26 in program WHARF.

CLEAR,GROUP COMMAND

The CLEAR,GROUP command (figure 5-3) allows you to remove group definitions. If the names parameter is coded as *, all groups are removed.

TABLE 5-1. CLEAR COMMAND VARIANTS

Command	Short Form	Function
CLEAR,AUXILIARY	CAUX	Closes the current auxiliary output file and clears all the auxiliary output options. This command can only be executed if standard output is currently defined.
CLEAR,BREAKPOINT	CB	Clears defined breakpoints.
CLEAR,GROUP	CG	Clears specified groups.
CLEAR,INTERPRET	CI	Turns off interpret mode. This command has the same effect as SET,INTERPRET,OFF.
CLEAR,OUTPUT	COUT	Turns off the standard output. This command can be executed only if an auxiliary output file is currently defined.
CLEAR,TRAP	CT	Clears specified traps.
CLEAR,VETO	CV	Turns off veto mode. This command has the same effect as SET,VETO,OFF.

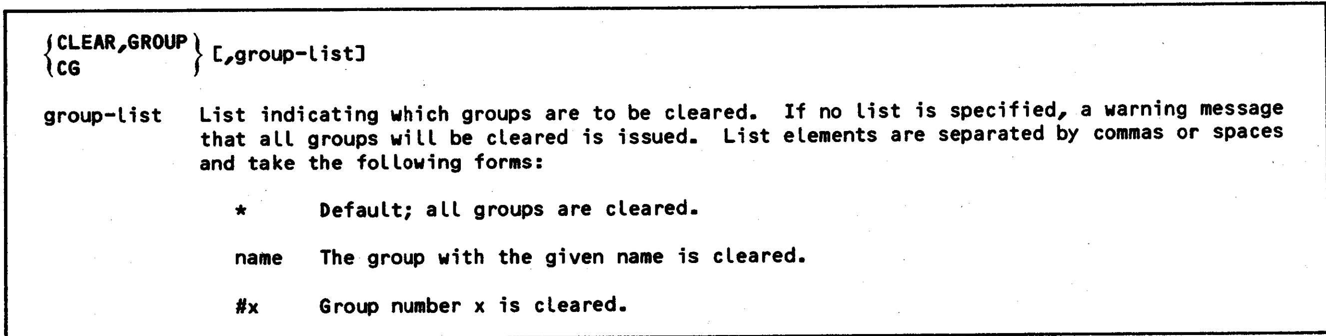


{CLEAR,BREAKPOINT} [,breakpoint-list]
{CB}

breakpoint-list   List indicating which breakpoints are to be cleared. If no list is specified, a warning message that all breakpoints will be cleared is issued. List elements must be separated by commas or spaces. Possible list elements and the breakpoints cleared are listed below.

*	Default; all breakpoints
(p,s)	All breakpoints in the specified overlay
program	All breakpoints in the specified program
location	The breakpoint at the specified location
address...address	All breakpoints in the specified address range
#x	Breakpoint number x

Figure 5-2. CLEAR,BREAKPOINT Command



{CLEAR,GROUP} [,group-list]
{CG}

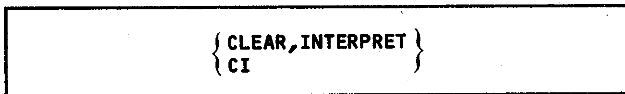
group-list   List indicating which groups are to be cleared. If no list is specified, a warning message that all groups will be cleared is issued. List elements are separated by commas or spaces and take the following forms:

*	Default; all groups are cleared.
name	The group with the given name is cleared.
#x	Group number x is cleared.

Figure 5-3. CLEAR,GROUP Command

CLEAR,INTERPRET COMMAND

The CLEAR,INTERPRET command (figure 5-4) turns off interpret mode. This command has the same effect as SET,INTERPRET,OFF described later in this section.

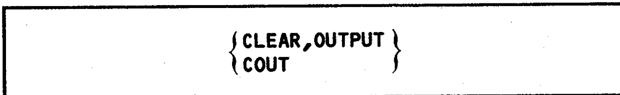


{CLEAR,INTERPRET}
{CI}

Figure 5-4. CLEAR,INTERPRET Command

CLEAR,OUTPUT COMMAND

The CLEAR,OUTPUT command (figure 5-5) turns off CID output to the standard output file. An auxiliary output file must be defined before the CLEAR,OUTPUT command can be executed (see SET,AUXILIARY command).



{CLEAR,OUTPUT}
{COUT}

Figure 5-5. CLEAR,OUTPUT Command

CLEAR,TRAP COMMAND

The CLEAR,TRAP command (figure 5-6) removes specified traps. If the trap-list is coded as *, all traps specified by type are removed. If type is coded as *, traps of all types are removed.

CLEAR,VETO COMMAND

The CLEAR,VETO command (figure 5-7) turns off veto mode. This command has the same effect as SET,VETO,OFF described later in this section.

```
{CLEAR,TRAP} ,type[,trap-list]
{CT        }
```

type — Type of traps to be cleared. * indicates all trap types.

trap-list — List indicating which traps of the specified type are to be cleared. If no list is specified, a warning message that all traps will be cleared is issued. List elements are separated by spaces or commas and take the following forms:

*	Default; all traps of the specified type are cleared.
(p,s)	All traps of the specified type in overlay (p,s) are cleared.
program	All traps of the specified type in the specified program module are cleared.
location	All traps of the specified type in the specified location are cleared.
address...address	All traps of the specified type in the specified address range are cleared.
#x	Trap #x is cleared.

Figure 5-6. CLEAR,TRAP Command

```
{CLEAR,VETO}
{CV        }
```

Figure 5-7. CLEAR,VETO Command

LIST COMMANDS

LIST commands allow you to display breakpoint, trap, and group names; command sequences; information about the program; and information about the state of CID itself. The LIST commands have abbreviated forms, such as LB for LIST,BREAKPOINT and LT for LIST,TRAP. The general form of a LIST command is:

LIST,object[,parameters]

where object is the particular object about which information is required, and the parameters indicate which particular objects to display. The LIST commands are shown in table 5-2. The term header refers to the parameter supplied with the SET command used to define a trap, breakpoint, or group, together with its associated number.

LIST,BREAKPOINT COMMAND

The LIST,BREAKPOINT command (figure 5-8) displays information relevant to breakpoints. The way in which the breakpoints are listed depends on the way in which the breakpoint-list elements are specified. If a list element is specified as an address range, such as a program, then the locations of any breakpoints in that range are displayed. If specific breakpoints are given, then the bodies of the breakpoints are listed as well.

If no breakpoint exists at the specified location, the following message is displayed:

NO BREAKPOINT location

An example of the LIST,BREAKPOINT command is:

LIST,BREAKPOINT,P.MASSON_20,P.ALMADEN

This command displays the header and body of the breakpoint at word 20 in program MASSON, and the locations of all breakpoints in program ALMADEN.

LIST,BREAKPOINT,#1,#4,#7

This command displays the headers and bodies (if any) of breakpoints 1, 4, and 7.

LIST,GROUP COMMAND

The LIST,GROUP command (figure 5-9) displays information relevant to groups. The command displays the definitions of those groups specified in the group-list. If the group-list is specified as *, then the names of all currently existing groups will be displayed.

If no group of the specified name exists, the following message is displayed:

NO GROUP group-name

An example of the LIST,GROUP command is:

LIST,GROUP,*; LIST,GROUP,SYNERGY

The first command displays the names of all currently defined groups; the second command displays the definitions and all the CID commands which make up the body of the group called SYNERGY.

TABLE 5-2. LIST COMMAND VARIANTS

Command	Short Form	Function
LIST,BREAKPOINT	LB	Lists the headers and bodies of specific breakpoints
LIST,GROUP	LG	Lists the names of all groups, or lists the bodies of specific groups
LIST,MAP	LM	Lists load map information about the program or about overlays in the program
LIST,STATUS	LS	Lists home program, terminal and auxiliary output options, and the state of interpret mode and veto mode
LIST,TRAP	LT	Lists the headers and bodies of specific traps
LIST,VALUES	LV	Lists the names and values of all symbols from a specified program

```
{LIST,BREAKPOINT} [,breakpoint-list]
{LB             }
```

breakpoint-list — List indicating which breakpoints are to be listed. Default is *. List elements are separated by commas or spaces and take the following forms:

*	The locations and frequency parameters of all breakpoints are listed.
(p,s)	The locations and frequency parameters of all breakpoints in overlay (p,s) are listed.
P.progname	The locations and frequency parameters of all breakpoints in the specified program are listed.
address...address	The headers and bodies of all breakpoints in the specified address range are listed.
location	The header and body of the breakpoint at the specified location is listed.
#x	The header and body of breakpoint #x are listed.

Figure 5-8. LIST,BREAKPOINT Command

```
{LIST,GROUP} [,group-list]
{LG        }
```

group-list — List indicating which groups are to be listed. List elements are separated by commas or spaces and take the following forms:

*	Default; the names of all groups are displayed.
group-name	The definition of the group with the specified name is displayed.
#x	The definition of group number x is displayed.

Figure 5-9. LIST,GROUP Command

LIST,MAP COMMAND

The LIST,MAP command (figure 5-10) displays load map information relating to programs, entry points, and overlays. LIST,MAP can be abbreviated to LM.

LIST,STATUS COMMAND

The LIST,STATUS command (figure 5-11) gives you information about the status of the CID environment. LIST,STATUS can be abbreviated to LS. The items displayed by the LIST,STATUS command are as follows:

- The name of the current home program
- The current output options for the CID standard output file
- The name of the current auxiliary output file and auxiliary output options
- The current state of the veto mode switch
- The current state of the interpret mode switch
- The number of breakpoints, traps, and groups currently defined

LIST,TRAP COMMAND

The LIST,TRAP command (figure 5-12) lists information relevant to traps. If the trap-list parameter specifies a list of locations or trap numbers, the command displays trap definitions. If the trap-list is specified as *, then all traps of the type specified by type are displayed. If type is specified as *, then all traps are displayed. LIST,TRAP can be abbreviated to LT.

The way in which the traps are listed depends upon the way in which the trap-list elements are specified. If a list element is specified as an area, such as a program, then the locations of any traps in that area are displayed. If specific traps are given, then the bodies of the traps are displayed.

```
{LIST,STATUS}
{LS         }
```

Figure 5-11. LIST,STATUS Command

If no traps exist at a specified place, the following message is displayed:

NO type TRAP place

Examples of the LIST,TRAP command are:

- LIST,TRAP,RJ,P.PRUNE_50...P.PRUNE_200

  This command displays the headers and bodies of all RJ traps (if any) from word 50 to word 200 in program PRUNE.

- LIST,TRAP,RJ,*

  This command displays the headers of all RJ traps that exist.

- LIST,TRAP,STORE,P.MACYS

  This command, where just the name of a program is given, will display the definitions of any trap of the specified type (in this case STORE) whose range lies within or encompasses the specified program. The same rule applies to common blocks and overlays.

LIST,VALUES COMMAND

The LIST,VALUES command (figure 5-13) lists source program values in specified BASIC, COBOL, and FORTRAN program modules compiled for use with CID. The setting of the home program has no effect on the LIST,VALUES command.

The LIST,VALUES command can generate a considerable quantity of output. You might not want all of the output to appear on the terminal, and can elect to send the output to the auxiliary output file by entering the SET,AUXILIARY command described in this section.

```
{LIST,MAP} [,place-list]
{LM      }
```

place-list List of places for which load map is displayed. Default is *. List elements are separated by commas or spaces and can take the following forms:

*	Names of all program modules and common blocks in the program are displayed. In an overlay environment, designations of overlays are listed and those currently loaded are flagged with an asterisk.
(p,s)	Names of all program modules and common blocks in overlay (p,s) are displayed.
P.progname	Origin, length, and all entry points in the specified program are displayed.

Figure 5-10. LIST,MAP Command

```
{LIST,TRAP} ,type[,trap-list]
{LT       }
```

type	Type of traps to be listed. * indicates all trap types.
trap-list	List indicating which traps of the specified type are to be listed. Default is *. List elements are separated by commas or spaces and take the following forms:

*	The scopes and types of all traps of the specified type are displayed.
(p,s)	The scopes and types of all traps of the specified type in overlay (p,s) are displayed.
P.progname	The scopes and types of all traps of the specified type in the specified program are displayed.
address...address	The scopes and types of all traps of the specified type in the specified address range are displayed.
location	The header and body of the trap in the specified location are displayed.
#x	The header and body of trap #x are displayed.

Figure 5-12. LIST,TRAP Command

```
{LIST,VALUES} [,place-list]
{LV         }
```

place-list List indicating which program modules are to have their values listed. List elements are separated by commas or spaces and take the following forms:

*	All values currently loaded are displayed.
(p,s)	All values in overlay (p,s) are displayed. Overlay (p,s) must be loaded.
progname	All values in the specified program are displayed. The specified program must be loaded.

Figure 5-13. LIST,VALUES Command

BASIC Program Modules

For each specified BASIC program module compiled for use wtih CID, the LIST,VALUES command lists all variable names in alphabetical order along with the current values of the variables. Values are formatted according to the variable type as declared in the BASIC program.

COBOL Program Modules

For each specified COBOL program module compiled for use with CID, the LIST,VALUES command lists all the data items in the same order that the data items are written in the data division of the COBOL source program. The entire data structures are shown in the listing, including group item name and the values of the elementary data items.

When tables are listed, the table name is given only for the first element. Subscripts are listed for all of the table elements.

Data items that share the same memory space with other data items (as the result of a REDEFINES clause) are not necessarily listed in a readable form. Changing the value of one data item changes the values of the other data items sharing the same space. The new values of the other data items might not be in a form that the LIST,VALUES command can interpret.

FORTRAN Program Modules

For each specified FORTRAN program module compiled for use with CID, the LIST,VALUES command lists all variable names in alphabetical order along with the current values of the variables. Values are formatted according to the variable type as declared in the FORTRAN program.

SAVE COMMANDS

SAVE commands copy the definitions of breakpoints, traps, or groups to a specified local file. The bodies of these definitions can be edited by a text editor while outside CID and then be replaced using the READ command. The general form of a SAVE command is:

SAVE,object,filename,list

The object is the type of object that is saved, the filename is the name of the file on which the definitions are copied, and the list indicates which particular objects are saved. The SAVE commands are listed in table 5-3.

TABLE 5-3. SAVE COMMAND VARIANTS

Command	Short Form	Function
SAVE,BREAKPOINT	SAVEB	Saves breakpoint definitions
SAVE,GROUP	SAVEG	Saves group definitions
SAVE,TRAP	SAVET	Saves trap definitions
SAVE,*	†	Saves breakpoint, group, and trap definitions

†No short form

The first line of each saved definition is the SET command used to create the object. The READ command can be entered to reestablish the definitions at a later time.

SAVE,BREAKPOINT COMMAND

The SAVE,BREAKPOINT command (figure 5-14) copies specified breakpoint definitions (including breakpoint bodies) to a local file. If the breakpoint-list is coded as *, then all breakpoints are saved.

SAVE,GROUP COMMAND

The SAVE,GROUP command (figure 5-15) copies specified group definitions to a file. If the group list is coded as *, then all groups are copied to the file.

SAVE,TRAP COMMAND

The SAVE,TRAP command (figure 5-16) copies specified trap definitions (including trap bodies) to a local file. If the trap-list is coded as *, then all traps are saved.

SAVE,* COMMAND

The SAVE,* command (figure 5-17) copies all breakpoint, trap, and group definitions to a local file.

SET COMMANDS

SET commands allow you to establish CID control objects such as breakpoints, traps, groups, and so forth. The general form of a SET command is:

SET,object,parameters

where object is the particular object that you wish to establish and parameters is a list of qualifiers associated with the particular object. The different SET commands are listed in table 5-4.

```
{SAVE,BREAKPOINT}
{SAVEB          } ,file-name[,breakpoint-list]

file-name        Logical file name of local file on which breakpoints are to be saved.

breakpoint-list  List indicating which breakpoints are to be saved. Default is *. List elements are
                 separated by commas or spaces and take the following forms:

                   *                  All breakpoints are saved.

                   (p,s)              All breakpoints in overlay (p,s) are saved.

                   P.progname         All breakpoints in the specified program are saved.

                   address...address  All breakpoints in the specified address range are saved.

                   location           All breakpoints at the specified location are saved.

                   #x                 Breakpoint #x is saved.
```

Figure 5-14. SAVE,BREAKPOINT Command

```
{SAVE,GROUP} ,file-name,group-list
{SAVEG     }
```

file-name   Logical file name of local file on which groups are to be saved.

group-list   List indicating which groups are to be saved. List elements are separated by commas or spaces and take the following forms:

- `*`   All groups are saved.
- `name`   The group with the specified name is saved.

Figure 5-15. SAVE,GROUP Command

```
{SAVE,TRAP} ,file-name,type[,trap-list]
{SAVET    }
```

file-name   Logical name of local file on which traps are to be saved.

type   Type of traps to be saved. * indicates all trap types.

trap-list   List indicating which traps of the specified type are to be saved. Default is *. List elements are separated by commas or spaces and take the following forms:

*	All traps of the specified type are saved.
(p,s)	All traps of the specified type in overlay (p,s) are saved.
P.progname	All traps in the specified program are saved.
address...address	All traps in the specified address range are saved.
location	All traps in the specified location are saved.
#x	Trap #x is saved.

Figure 5-16. SAVE,TRAP Command

```
SAVE,*,file-name
```

file-name   Logical name of local file on which breakpoints, groups, and traps are to be saved.

Figure 5-17. SAVE,* Command

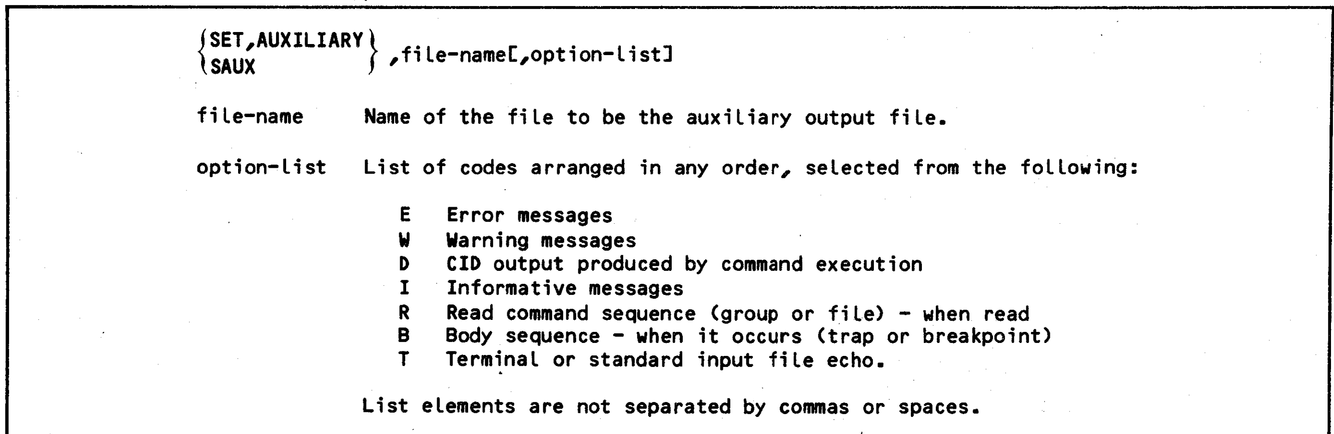
These language-independent SET commands do not include the COBOL CID SET command described in section 6. To distinguish one of these SET commands from the COBOL CID SET command, these commands must be entered in short form or with a comma following the command name when the home program is a COBOL program compiled for use with CID.

TABLE 5-4. SET COMMAND VARIANTS

Command	Short Form	Function
SET,AUXILIARY	SAUX	Establishes file name and output options for the auxiliary output file
SET,BREAKPOINT	SB	Establishes a breakpoint
SET,GROUP	SG	Establishes a group
SET,HOME	SH	Establishes the name of the home program
SET,INTERPRET	SI	Turns interpret mode on or off
SET,OUTPUT	SOUT	Establishes output options for the standard output file
SET,TRAP	ST	Establishes a trap
SET,VETO	SV	Turns veto mode on or off

SET,AUXILIARY COMMAND

The SET,AUXILIARY command (figure 5-18) allows you to define an optional local auxiliary output file which can be used as a log of the debug session. You would most likely dispose this file either to the terminal or to a line printer.



```
{SET,AUXILIARY} ,file-name[,option-list]
{SAUX         }

file-name     Name of the file to be the auxiliary output file.

option-list   List of codes arranged in any order, selected from the following:

                 E   Error messages
                 W   Warning messages
                 D   CID output produced by command execution
                 I   Informative messages
                 R   Read command sequence (group or file) - when read
                 B   Body sequence - when it occurs (trap or breakpoint)
                 T   Terminal or standard input file echo.

              List elements are not separated by commas or spaces.
```

Figure 5-18. SET,AUXILIARY Command

The options in this command determine the type of output written to the auxiliary file. Each SET,AUXILIARY command supersedes any previous SET,AUXILIARY command.

You can use the SET,AUXILIARY command in conjunction with the SET,OUTPUT command as follows: Select options in the SET,AUXILIARY command such that large quantities of output are stored on the auxiliary output file; select options in the SET,OUTPUT command such that brief messages are output to the terminal. Whenever you need information from the auxiliary file, suspend the debug session (see the SUSPEND command, later in this section), look at the auxiliary file using a system editor, and resume the session (see the DEBUG control statement, section 2).

It is possible to eliminate all terminal output by issuing a CLEAR,OUTPUT command. This can only be done, however, if an auxiliary file has been defined with at least the E option specified.

Only one auxiliary output file can be in use at a time. Thus, if a subsequent SET,AUXILIARY command is issued that references a file name different from the file name referenced in the previous SET,AUXILIARY command, a warning message is issued. If you respond with a positive acknowledgment (ACCEPT, YES or OK), the old auxiliary output file is closed before the new one is established. To avoid the warning messages, you must first issue a CLEAR,AUXILIARY command (described earlier in this section). If a subsequent SET,AUXILIARY command is issued merely to change output options, but retains the same file name, no warning message is issued, and no closing of the file occurs.

Prior to the first SET,AUXILIARY command, as well as after a CLEAR,AUXILIARY command, no auxiliary file is defined.

Disposition of the auxiliary output file is your responsibility; the file is not printed by CID. You should ensure that the file chosen for auxiliary output is not one that will be read, written, or otherwise manipulated by the program.

The E option must not be omitted on both the standard and auxiliary output files. Error messages must be output to at least one of these files.

SET,BREAKPOINT COMMAND

A breakpoint (see section 3) is established by means of the SET,BREAKPOINT command (figure 5-19). The location parameter in the SET,BREAKPOINT command is an address. In BASIC, COBOL, and FORTRAN programs compiled for use with CID, source language symbols, such as line numbers, should usually be specified (see section 4).

Specifying a left bracket ([) in the SET,BREAKPOINT command activates collect mode. When collect mode is activated, you can enter a breakpoint body that executes automatically when execution reaches the breakpoint location. Breakpoint bodies are a form of command sequence and are described in section 3.

When a breakpoint is set, CID assigns the breakpoint a number in the range 1 through 16. This breakpoint number, referred to in the form #n, provides a convenient way of referring to breakpoints in LIST or CLEAR commands. It is also used in the breakpoint reporting message.

When a breakpoint is reached and the criteria as determined by the frequency parameters are met, then the breakpoint is honored.

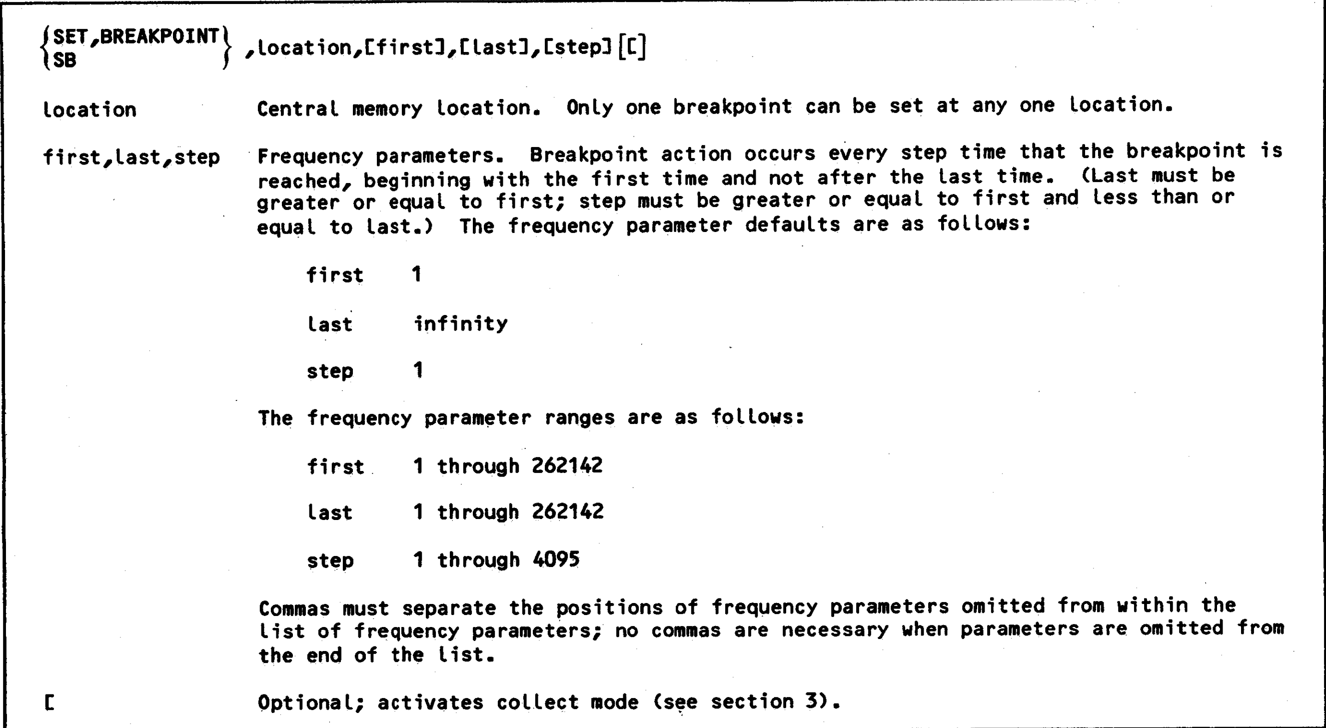
SET,GROUP COMMAND

The SET,GROUP command (figure 5-20) establishes a group. The opening left bracket is optional but included in the syntax for compatibility with trap and breakpoint SET commands with bodies. The closing right bracket is required.

A group is a type of command sequence (see section 3). Upon establishment of a group, a number is assigned by CID. This group number, referred to in the form #n, provides a convenient way of referring to groups in the LIST or CLEAR commands described in this section.

SET,HOME COMMAND

The SET,HOME command (figure 5-21) designates a specific program module as the home program. The home program is described in section 3.



{SET,BREAKPOINT} ,location,[first],[last],[step] [C]
{SB}

location — Central memory location. Only one breakpoint can be set at any one location.

first,last,step — Frequency parameters. Breakpoint action occurs every step time that the breakpoint is reached, beginning with the first time and not after the last time. (Last must be greater or equal to first; step must be greater or equal to first and less than or equal to last.) The frequency parameter defaults are as follows:

first	1
last	infinity
step	1

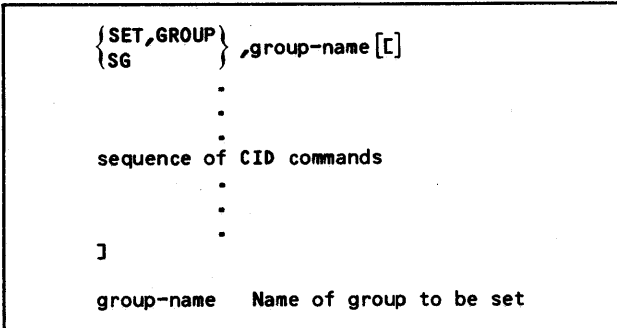
The frequency parameter ranges are as follows:

first	1 through 262142
last	1 through 262142
step	1 through 4095

Commas must separate the positions of frequency parameters omitted from within the list of frequency parameters; no commas are necessary when parameters are omitted from the end of the list.

C — Optional; activates collect mode (see section 3).

Figure 5-19. SET,BREAKPOINT Command



{SET,GROUP} ,group-name [C]
{SG}

.
.
.

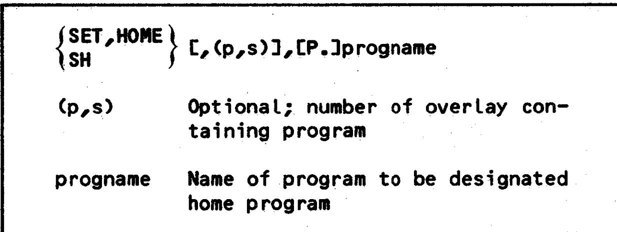
sequence of CID commands

.
.
.

]

group-name — Name of group to be set

Figure 5-20. SET,GROUP Command



{SET,HOME} [,(p,s)],[P.]progname
{SH}

(p,s) — Optional; number of overlay containing program

progname — Name of program to be designated home program

Figure 5-21. SET,HOME Command

The command is useful if a series of references is to be made to some program other than the current program module. By designating the program module to be referenced as the home program, such references can be made in the implicit home program form _n, as described in section 4.

If BASIC, COBOL, or FORTRAN CID commands are to be used to reference source symbols not in the current home program, then SET,HOME must first be entered to change the home program designation to the one in which the symbols occur.

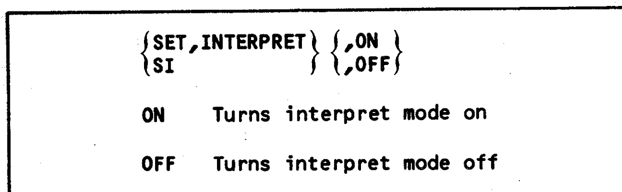
When SET,HOME is executed, the program module indicated is made the home program. The designated program module can be in an overlay not currently loaded. Issuing a SET,HOME command in this case, however, would only be useful if SET,BREAKPOINT or SET,TRAP commands specifying locations within such a program were to be issued; all other references to unloaded program locations are not allowed.

A designated program remains the home program until it is changed either explicitly by the SET,HOME command, or by a trap or breakpoint occurring in some other program.

SET,INTERPRET COMMAND

The SET,INTERPRET command (figure 5-22) allows you to explicitly control the use of interpret mode. Interpret mode is turned on or off as indicated by the command parameter. An informative message is issued in each case. When interpret mode is turned off, traps that depend on interpret mode do not occur.

Interpret mode is also turned on when any RJ, XJ, JUMP, STORE, FETCH, or INSTRUCTION trap is established. If, after setting such traps, a SET,INTERPRET,OFF or CLEAR INTERPRET command is issued, the traps are rendered inoperative although still defined. If a SET,INTERPRET,ON is subsequently issued, the traps become operative again.



{SET,INTERPRET} {,ON}
{SI} {,OFF}

ON Turns interpret mode on

OFF Turns interpret mode off

Figure 5-22. SET,INTERPRET Command

SET,OUTPUT COMMAND

The SET,OUTPUT command (figure 5-23) allows you to control the kinds of CID output that are written to the CID standard output file. The options designated in the option list determine the types of output written to this file. Each SET,OUTPUT command supersedes any previous SET,OUTPUT command.

When the list is omitted, the default options are E, W, D, and I.

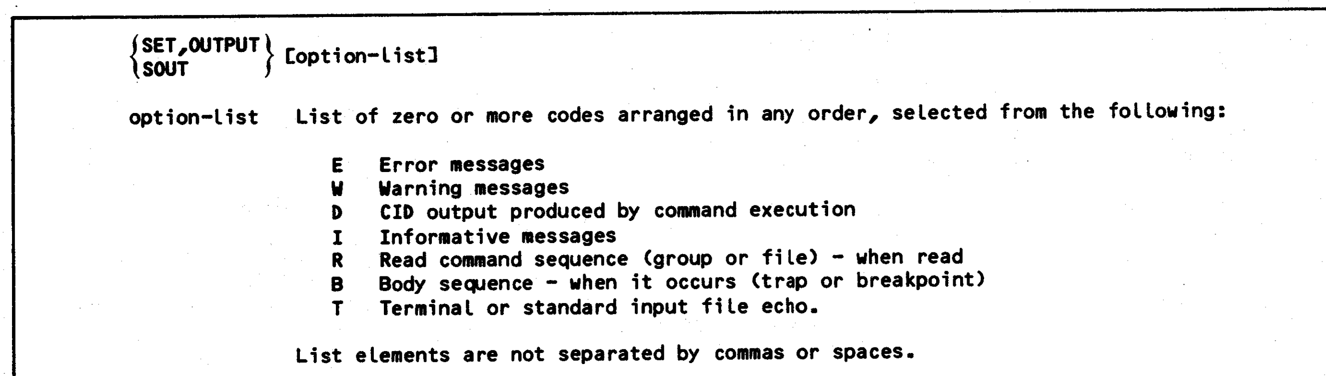
If the E option is omitted, error messages are suppressed, response mode (see section 7) is not entered when an error occurs, and erroneous commands are skipped. In this case, an auxiliary file must be defined with the E option specified.

If the W option is omitted, warning messages are suppressed, response mode is not entered, and commands which normally would result in warning messages are executed. Trap and breakpoint report messages are not suppressed when the I option is omitted.

SET,TRAP COMMAND

Traps are established by the SET,TRAP command (figure 5-24). The command indicates the type of trap to be established; the types are shown in table 5-5. Trap types are further described in section 3. The SET,TRAP command also indicates the scope of the trap. The way in which the scope is indicated depends on the type of trap. For an OVERLAY trap, the scope is a parenthesized pair of overlay numbers. For all other trap types, the scope is an address range. An execution address range is a range of addresses in executable code within which any event of the specified type is to cause a trap. For FETCH and STORE, the address range indicates not a section of executable code but a section of data words within which any fetches or stores are to be monitored. In other words, if a fetch from or a store into any word in the specified range is executed, a trap occurs. An asterisk (*) indicates unrestricted scope; the condition is trapped anywhere in the program.

When a trap is set, the trap is assigned a number in the range 1 through 16. This trap number, referred to in the form #n, provides a convenient way of referring to traps in the LIST or CLEAR commands described in this section. The trap number is also used in the trap reporting message described in section 3. A trap remains established for the remainder of the debug session unless the trap is redefined by another SET,TRAP command or cleared by a CLEAR,TRAP command.



{SET,OUTPUT} [option-list]
{SOUT}

option-list List of zero or more codes arranged in any order, selected from the following:

E Error messages
W Warning messages
D CID output produced by command execution
I Informative messages
R Read command sequence (group or file) - when read
B Body sequence - when it occurs (trap or breakpoint)
T Terminal or standard input file echo.

List elements are not separated by commas or spaces.

Figure 5-23. SET,OUTPUT Command

```
{SET,TRAP} ,type,scope[,report level][C]
{ST      }
```

type — Trap type or abbreviation of trap type. Indicates condition causing suspension of execution.

scope — Portion of user field length to which trap is to apply. Must be in valid format for trap type. An asterisk (*) indicates unrestricted scope.

report level — Optional; specifies form in which location of trap is to be reported. The report level is one of the following:

- L — Line number (FORTRAN, BASIC, and COBOL). Compiler-assigned line number, as shown in source listing. Default for FORTRAN, BASIC, and COBOL programs compiled in debug mode.
- P — Program address. Default for programs other than FORTRAN, BASIC, and COBOL programs compiled in debug mode.
- PR — Procedure name (COBOL).
- S — Statement label (FORTRAN and BASIC).

C — Optional; activates collect mode (section 3).

Figure 5-24. SET,TRAP Command

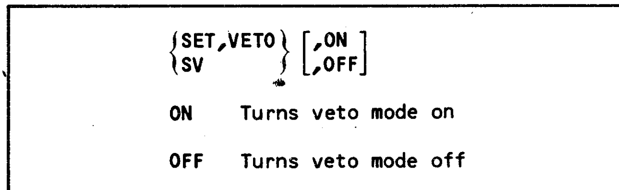
TABLE 5-5. TRAP TYPES

Short Form	Trap Type	Condition	Scope	CID Gets Control
A	ABORT	Abnormal termination except operator drop or kill	Execution address range	After
E	END	Normal program termination	Execution address range	After
F	FETCH†	Fetch from memory	Fetch address range	After
I	INSTRUCTION†	Beginning of any machine instruction	Execution address range	Before
INT	INTERRUPT	User interrupt	Execution address range	After
J	JUMP†	Jump instruction other than return jump or exchange jump (if jump takes place)	Execution address range	Before
L	LINE	Beginning of an executable source line of a BASIC, COBOL, or FORTRAN program compiled for use with CID	Execution address range	Before
OVL	OVERLAY	Overlay load	Overlay number	After
PROC	PROCEDURE	Beginning of a procedure in a COBOL program compiled for use with CID	Execution address range	Before
	RJ†	Return jump instruction entry and return	Execution address range	Before
S	STORE†	Store to memory	Store address range	After
	XJ†	Central exchange jump	Execution address range	Before

†Turns on interpret mode.

SET,VETO COMMAND

The SET,VETO command (figure 5-25) turns veto mode on or off. The CLEAR,VETO command has the same effect as SET,VETO,OFF. Veto mode is described in section 7.



{SET,VETO | SV} [,ON | ,OFF]

ON   Turns veto mode on

OFF   Turns veto mode off

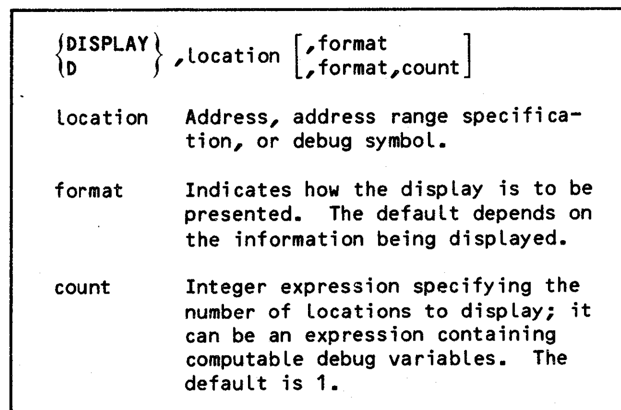
Figure 5-25. SET,VETO Command

OTHER LANGUAGE-INDEPENDENT COMMANDS

The following paragraphs describe other language-independent CID commands. The commands are listed alphabetically.

DISPLAY COMMAND

The DISPLAY command (figure 5-26) displays the contents of one or more central or extended memory locations. This language-independent DISPLAY command is not the same as the COBOL CID DISPLAY command described in section 6. To distinguish this command from the COBOL CID DISPLAY command, this command must be entered in short form or with a comma following the command name when the home program is a COBOL program compiled for use with CID.



{DISPLAY | D} ,location [,format | ,format,count]

location   Address, address range specification, or debug symbol.

format   Indicates how the display is to be presented. The default depends on the information being displayed.

count   Integer expression specifying the number of locations to display; it can be an expression containing computable debug variables. The default is 1.

Figure 5-26. DISPLAY Command

This command displays memory words without regard to data structures. If you are displaying locations other than debug variables, you should have an understanding of the data structures being displayed when you use this command. Language-dependent commands are generally more useful when displaying values from BASIC, COBOL, and FORTRAN programs compiled for use with CID.

If the location parameter is a memory address, the contents of the number of words specified by the count parameter beginning with that address are displayed. If location is an address range specification, then the contents of the entire address range are displayed and the count is ignored. If location is a debug variable, the value of the debug variable is displayed.

Format codes are shown in table 5-6. The default format for displaying debug variables is octal, except for the following:

#P,#EA	Symbolic address (A)
#HOME,#LINE,#PROC	Character (C)
#BP,#TP,#GP,#INSL, #V1 through #V10, #ERRCODE,#CPUERR	Decimal integer (I)

TABLE 5-6. DISPLAY FORMATS

Format Code	Format Displayed
A	Symbolic address
C	Character data: that is, Hollerith or string data
D	Double-precision floating-point
F	Floating-point
I	Signed 60-bit decimal integer
O	Unsigned 60-bit octal number, indicated by the letter O

If the A format is specified, only the lower 18 bits of specified central memory locations or the lower 24 bits of extended memory locations are used as the address which is then displayed in symbolic form.

The default format for the DISPLAY command is as follows when the location parameter is expressed as a variable name from a BASIC or FORTRAN program compiled for use with CID:

I If the variable type is signed 60-bit integer

C If the variable type is character

F If the variable type is floating-point

D If the variable type is double-precision

If the variable type is not signed 60-bit integer, character, floating-point, or double-precision, the default format is octal.

The default format for all other types of location parameters (including locations in COBOL programs compiled for use with CID) is octal.

DISPLAY must not reference locations in an unloaded overlay or in CID itself.

Examples of the DISPLAY command are:

- ?DISPLAY,2
  2=32323 23232 17125 01046

- ?DISPLAY,2...12,F
  2=.35567145154957E+21 0.0 0.0 0.0 0.0 0.0 0.0
  " +8 = 0.0 0.0 0.0

- ?DISPLAY,2,D
  2=.35567145154956821382667207 4D+21

- ?DISPLAY,#P
  #P=P.SUB1_23B

- ?DISPLAY,#P,O
  #P=021171

- ?DISPLAY,#HOME
  #HOME=P.SUB1

ENTER COMMAND

The ENTER command (figure 5-27) assigns a value to one or more words in memory or to a debug variable. You should have an understanding of the data structures being changed before entering this command when the location parameter is not a debug variable.

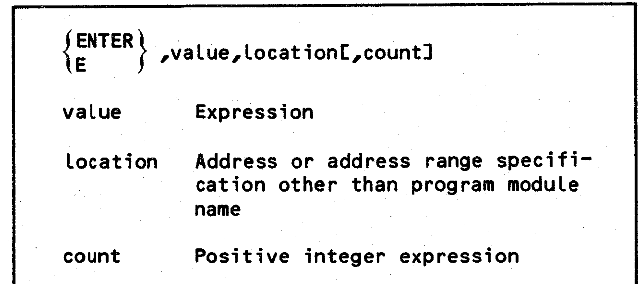


Figure 5-27. ENTER Command

If the location parameter is an address, the value supplied is stored at the number of consecutive words indicated by the count parameter beginning with that address. The default for count is 1. If location is an address range specification, then the specified value is stored at all addresses in the range and count is ignored.

ENTER cannot reference locations in an unloaded overlay or in CID itself.

Examples of the ENTER command are:

- ENTER,0,C.BLKA_5

  The value zero is entered at the sixth location of common block BLKA. A default value of 1 is used for count.

- ENTER,0,C.BLKA_0...C.BLKA_3

  Zeros are entered into the first four locations of BLKA.

- ENTER,#P,#V1

  The value of #P is entered into #V1.

- ENTER,P.PROG,14,C.BLKB_23

  The address of the 15th location in PROG is entered into the 24th word of BLKB.

- ENTER,!PROG_14,C.BLKB_0,24

  The contents of the 15th location in PROG are entered in the first 24 words of BLKB.

EXECUTE COMMAND

The EXECUTE command (figure 5-28) starts or resumes program execution. Program execution is initiated or resumed at the location indicated. If no location is specified, execution is resumed at the location where execution was suspended or is initiated at the start of the program.

```
{EXECUTE} [,location]
{EXEC   }

location   Optional; if given, must be an
           address within a program module.
           Default is the current location
           #P.
```

Figure 5-28. EXECUTE Command

Care should be taken when you specify a location in the EXECUTE command, because this action changes the flow of program execution. When the EXECUTE command is entered to start program execution at the beginning of a debug session, the EXECUTE command should be entered with no parameters; if the location parameter is specified when the EXECUTE command is entered to start program execution, program initialization does not take place.

The EXECUTE command with no parameters is not allowed after an ABORT or END trap with no reprieve code, or after reprieve code is completed. In these cases, the only allowable form of the command is EXECUTE, location, where location is other than #P.

In cases where no current command sequence is suspended, the GO command causes the same action as EXECUTE.

GO COMMAND

The GO command (figure 5-29) causes an exit from the current sequence or mode and a resumption of suspended processing. A new location for resumption of program execution can be specified via the optional location parameter.

```
GO[,location]

location   Optional; new location for resump-
           tion of execution.
```

Figure 5-29. GO Command

When issued from within a breakpoint or trap body, GO causes resumption of program execution either at the place where execution was suspended, or at the location specified. In this case, action is identical to the EXECUTE command. Care should be taken when you specify a location, because this action changes the flow of program execution.

When issued from within the body of a group or file, GO causes an exit from that sequence and a resumption of the process that was active when the READ command that invoked the current sequence was issued. If the previously active process was itself a command sequence, control is transferred to the command following the invoking READ command. If the READ command was issued from the terminal, interactive mode is resumed.

When issued in interactive mode as a result of a PAUSE command within a command sequence, GO causes a resumption of that suspended sequence at the command following the PAUSE command.

When issued in interactive mode as a result of an implicit PAUSE (for example, the start of the debug session, or after a breakpoint or trap having no body), GO causes a resumption of program execution. In this case, action is identical to the EXECUTE command. Care should be taken when you specify the location in the GO command, because this action changes the flow of program execution. When the GO command is used at the start of a debug session, the location should not be specified; if the location is specified at the start of a debug session, program initialization does not take place.

HELP COMMAND

The HELP command (figure 5-30) can provide immediate information about CID features or commands. It is not a substitute for a reference manual, but rather acts as an on-line summary that can be accessed selectively.

HELP issued with no parameters or * lists a set of permissible parameter values and a corresponding subject category for each value. HELP,subject can then be entered to get more detailed information about the subject. For example, HELP,CMDS lists the commands and gives a brief explanation of each.

```
HELP [,*            ]
     [,subject      ]
     [,command-name ]

*              Default; displays subject
               index

subject        Displays command index for
               subject

command-name   Displays information about
               specified command
```

Figure 5-30. HELP Command

For more detailed information regarding a specific command, HELP can be entered with the command name as the parameter value.

JUMP COMMAND

The JUMP command (figure 5-31) transfers control to a specific label within a command sequence. Control can be transferred either forward or backward.

```
JUMP,label

label   Location of a command as defined in
        a LABEL Command
```

Figure 5-31. JUMP Command

The command sequence is searched for the label in a forward direction from the position of the JUMP command. The search continues until either the label is found, or the end of the sequence or file is found.

If the label is not found, the search continues at the start of the sequence and proceeds in a forward direction until either the label is found or the same JUMP command is reached.

If the label is found, execution of commands continues from that point in the command sequence. If the label is not found, an error message is issued and you are prompted for response.

For a label to be found by a JUMP command, the labeled command must be the first command on a line.

No check is made for duplicate labels in a given command sequence. Transfer of control is made to the label that is encountered first.

In interactive mode, outside of command sequences, jumps can only be made to the beginning of the line in which the JUMP command is entered. In batch mode, jumps within the standard input file DBUGIN are permissible.

LABEL COMMAND

The LABEL command (figure 5-32) defines a label for the location in the command sequence where the LABEL statement occurs. JUMP commands appearing elsewhere in the same command sequence can reference the label.

```
LABEL,label

label   Character string not more than seven
        characters in length and consisting
        of any arrangement of letters and
        digits
```

Figure 5-32. LABEL Command

Labels are local to the command sequence in which they are defined, so the same label can be used in many different command sequences without conflict. No check is made for duplicate labels in a given command sequence.

For a label to be found by a JUMP command, the labeled command must be the first command on a line. In interactive mode, a label is meaningful only to JUMP commands that are in the same line.

MESSAGE COMMAND

The MESSAGE command (figure 5-33) allows a message to be issued while executing a command sequence. The message text supplied is issued to the standard output file (usually the terminal).

MOVE COMMAND

The MOVE command (figure 5-34) assigns the values contained at a source set of memory locations to a destination set of locations. The source and destination locations are specified as address ranges or as single, beginning locations. The MOVE command assigns words of data beginning and ending at the addresses specified without regard to the data structures involved. You should have an understanding of the data structures being changed before entering this command.

```
MESSAGE,'message text'

message text   Any string of characters. The string can include ; and ]. The message text is delimited
               by either a pair of apostrophes (') or by a pair of quotes ("). Quotes or apostrophes
               identical to those used to delimit the text are indicated within the text by two consecu-
               tive occurrences of the character.
```

Figure 5-33. MESSAGE Command

```
{MOVE} ,source,destination[,count]
{M   }

source        Address expression or address
              range specification other than
              program name

destination   Address expression or address
              range specification other than
              program name

count         Integer expression; default is
              one
```

Figure 5-34. MOVE Command

This language-independent MOVE command is not the same as the COBOL CID MOVE command described in section 6. To distinguish this command from the COBOL CID MOVE command, this command should be entered in short form or with a comma following the command name when the home program is a COBOL program compiled for use with CID.

The action taken when the MOVE command is executed depends on whether either the source or destination parameter is a range specification. If the source or destination is a range specification, the count parameter is ignored even if present and enough words are moved from the source to fill the destination range. If the source range is smaller than the destination range, the words in the source range are moved repeatedly (in the same order) until the destination range is filled. If the source range is larger than the destination range, only as many words as are needed are moved.

If neither source nor destination is a range specification, the count parameter determines the number of words to be moved. The data in the source is moved to destination, source+1 to destination+1, and so forth, until source+(count-1) is moved to destination+(count-1). The default value of count is 1.

MOVE must not reference locations in an unloaded overlay or in CID itself.

An example of the MOVE command is:

```
MOVE,C.BLKA_0...C.BLKA_2,C._0...C._6
```

The first three words of common block BLKA are moved repeatedly until values are sent to the first seven locations of the unlabeled common block. The destination, when completed, contains the following:

Location	Value
C._0	!C.BLKA_0
C._1	!C.BLKA_1
C._2	!C.BLKA_2
C._3	!C.BLKA_0
C._4	!C.BLKA_1
C._5	!C.BLKA_2
C._6	!C.BLKA_0

An example of the MOVE command where the destination range is shorter than the source range is:

```
MOVE,C.BLKA_0...C.BLKA_5,C._0...C._3
```

Since the destination is only four words long, only the first four words of BLKA are moved into the first four words of the unlabeled common block.

If a count parameter is supplied in either of the preceding examples, it is ignored.

An example of the MOVE command using the count parameter is:

```
MOVE,C.BLKA_0,C._0,4
```

Here, neither the source (C.BLKA_0) nor the destination (C._0) is an address range specification; the count parameter is used and results are identical to the previous example.

Another example of the MOVE command is:

```
MOVE,C.BLKA_0,C._0
```

The first word of BLKA is moved to the first word of unlabeled common. Since no count is provided, only one word is moved.

NULL COMMAND

The NULL command (figure 5-35) is a do-nothing command. That is, when executed, no action occurs. Its main purpose is as a replacement command when responding to an error, warning, or veto (see section 7).

```
{NULL}
{†   }
```

†The short form of the NULL command is an empty line (or an empty area between semicolons).

Figure 5-35. NULL Command

PAUSE COMMAND

The PAUSE command (figure 5-36) is used within a command sequence to cause suspension of the automatically executing command sequence and to place the system in interactive mode so that CID commands can be entered directly from the terminal. Interactive mode is signaled by the question mark prompt for terminal input.

On the occurrence of the first PAUSE command in a sequence body of a trap or breakpoint, the appropriate trap or breakpoint report is issued prior to entering interactive mode.

```
PAUSE[,'message-text']

message-text   Any string of characters. The string can include ; and ]. The message text is delimited
               by either a pair of apostrophes (') or by a pair of quotes ("). Quotes or apostrophes
               identical to those used to delimit the text are indicated within the text by two consecu-
               tive occurrences of the character.
```

Figure 5-36. PAUSE Command

If automatic mode of execution was initiated from interactive mode by executing a group or file with the READ command, then the PAUSE command does not result in any automatic message being produced.

In any case where the reason for the PAUSE might not be clear when it is executed, you should use the second form of the command with a suitable message text. The message text is issued after any trap or breakpoint report, and immediately prior to entering interactive mode.

The PAUSE command has no effect when it is executed outside of a command sequence.

QUIT COMMAND

The QUIT command (figure 5-37) terminates a debug session.

```
     ┌,NORMAL┐
QUIT │,N     │
     │,ABORT │
     └,A     ┘

NORMAL or N    Default; normal termination of
               debug session occurs.

ABORT or A     Abort termination of debug
               session occurs.
```

Figure 5-37. QUIT Command

The current debug session is terminated; that is, control is returned to the operating system to process the next control statement. If the parameter value supplied is NORMAL (or implied by default), normal termination occurs.

Files used by BASIC, COBOL, and FORTRAN programs are automatically closed by the QUIT command. Files used by other programs are not automatically closed unless a subroutine that closes the files has entry point SYSEND and has been loaded with the program.

If the parameter value supplied is ABORT, an abort type of termination occurs. In this case, system action causes the message DEBUG ABORTED to be inserted into the dayfile. If CID is executed in batch mode, then subsequent control statements in the control statement section are skipped until an EXIT statement or the end of the job is encountered. If an EXIT control statement is encountered, control statement execution then resumes with the first statement following the EXIT statement.

READ COMMAND

The READ command (figure 5-38) is used in the following ways:

- To process CID commands stored on a file by some facility not provided by CID itself (for example, the editor)
- To reconstitute breakpoint, trap, and group definitions previously saved on a file
- To invoke the command sequence of a group

```
READ {,file-name }
     {,group-name}

file-name    Name of file containing commands
             to be executed

group-name   Name of group to be executed
```

Figure 5-38. READ Command

When READ is executed, a search is made for a group with the specified name. If a group is found, then commands are read and executed from the group. If there is no group with that name, a search is made for a file with that name. If a file is found, the file is rewound, commands are read and executed, and the file remains at end-of-information. If no such group or file is found, an error message is issued and a response awaited.

A command sequence being read can itself contain a READ command. READ commands can be nested; however, files or groups cannot be read recursively. That is, the reading of a file or group must be ended before it can be read again.

Before a file can be read, it must be assigned to the job. This is accomplished through control statements appropriate to the operating system being used. These statements must be issued while you are in system command mode, either prior to entering CID, or upon issuing a SUSPEND command.

SKIPIF COMMAND

The SKIPIF command (figure 5-39) conditionally skips commands when a specified relation is satisfied. With SKIPIF, trap bodies can be written which cause action dependent on where the trap occurred, or bodies can be made to act according to the values of program or debug variables.

```
SKIPIF,value1,relation,value2

value_i     Integer expression

relation    One of the following:

               EQ   Equal
               NE   Not equal
               LT   Less than
               LE   Less than or equal
               GE   Greater than or equal
               GT   Greater than
```

Figure 5-39. SKIPIF Command

The logical relationship is evaluated, and, if the result is true, the next command in the command sequence is skipped. If the relationship is false, the next command in the command sequence is executed. By using a JUMP following a SKIPIF, any number of commands can be skipped.

STEP COMMAND

The STEP command (figure 5-40) initiates or resumes program execution until a specified number of lines or COBOL procedure names have been reached. CID suspends program execution and issues the following message:

```
*S type AT address
```

The address is a reference to the next line or instruction to be executed.

When stepping a number of lines, the STEP command counts only lines in BASIC, COBOL, and FORTRAN programs compiled for use with CID. Nonexecutable lines and lines that are continued from a previous line are not counted. COBOL procedure-name lines are counted.

When stepping a number of procedures, the STEP command counts only procedure-name lines in COBOL programs compiled for use with CID.

Whenever you gain control before a STEP command has completed its action, the STEP command is discontinued. For example, if you issue the command:

```
STEP,5,LINES
```

and a breakpoint occurs after two lines are executed, you gain control at the breakpoint, and the STEP command is discontinued.

If a STEP command causes you to gain control at the same time as another event causes you to gain control, the other event gives you control, and the STEP command is discontinued.

At a breakpoint or trap where a command sequence is executed and you do not gain control, the STEP command remains in effect. If, however, the command sequence causes you to gain control (as the result of a PAUSE command, for example), then the STEP command is discontinued.

If you gain control as the result of an error or warning in a command sequence, the STEP command is not discontinued. When an error is reported, you can only respond to that error; you do not have general control.

{STEP | S} ,[n],[type],[scope]

n — Number of lines or procedure names counted before execution is suspended. This value must be less than 536870911.

type — LINE, LINES, or L if lines are to be stepped. PROCEDURE, PROCEDURES, PROC, PROCS, or PR if procedure-names are to be stepped.

scope — Address range indicating within which area program lines or procedure names are to be counted. The scope parameter can take one of the following forms:

*	All lines or procedure names are to be counted.
location1...location2	All lines in the specified address range are to be counted.
P.progname	All lines in the specified program are to be counted.

In this command, commas must separate the positions of parameters omitted from the middle of the list. No commas are necessary when parameters are omitted from the end of the list.

Defaults are as follows: If no parameters are entered, the previous STEP command is reexecuted. (If no previous STEP command has been executed during the debug session, STEP,1,LINE,* is executed.) If at least one parameter is entered, the defaults are n=1, type=LINE, and scope=*.

Figure 5-40. STEP Command

SUSPEND COMMAND

The SUSPEND command (figure 5-41) provides a means of leaving CID and returning to the operating system command mode. With SUSPEND, the option is open to return later to CID with the program, environment, and status restored to what it was when the SUSPEND command was issued. You can use this capability to edit command sequences (see section 3).

```
SUSPEND[,file-name]

file-name   Local file on which the current
            CID environment is saved.  The
            default name of ZZZZZDS is used
            if no file name is specified.
```

Figure 5-41. SUSPEND Command

The current CID environment (breakpoint, trap, and group definition), current status (interpret and veto mode settings, output options, and home program designation), debug variables and tables, as well as the program, are saved on a local file. A return is then made to the operating system command mode.

A warning message is issued if a previous version of the file saved by SUSPEND exists. This would be the case if a SUSPEND issued earlier in the same terminal session had not been resumed. A positive response to the warning message results in execution of the new SUSPEND which overwrites the file created by the prior SUSPEND, thus making resumption from the prior suspension impossible.

No closing or saving of your files is performed when the SUSPEND command is executed. Note that unclosed files might become unusable or incomplete if the debug session is not resumed.

The CID status, CID environment, and program are restored when the DEBUG(RESUME) control statement is executed. Resuming the debug session does not restore the position and status of your program files to their state at CID suspension. Thus, while the debug session is suspended, operations should not be performed that would alter the position or status of these files.

The suspended debug session is saved on a local file. If you want to resume the session in a future terminal session or job, you must make the local file permanent.

TRACEBACK COMMAND

The TRACEBACK command (figure 5-42) provides, in concise form, a high level summary of program flow leading to the most recent execution of the program module specified or implied.

```
TRACEBACK [,E.entrypoint]
          [,P.progname  ]

no parameter   Home program is designated
               program.

entrypoint     Program containing entry point
               is designated program.

progname       Named program is designated
               program.
```

Figure 5-42. TRACEBACK Command

The TRACEBACK command produces a list of program module names, beginning with the designated program and progressing backward through successive levels. At each level, it displays the name of the program that last called that module and the location within the module where the call took place.

If the program specified in the TRACEBACK command has not been called during the debug session, CID issues the error message PROGRAM progname NOT CALLED, where progname is the name of the program. If a program module contains more than one entry point, a warning message is issued and the first entry point is chosen if the response is affirmative (YES, OK, or ACCEPT).

Examples of the TRACEBACK command are:

- ? TRACEBACK
  P.SUB CALLED FROM P.TEST_217B

- ? TRACEBACK,E.ABC
  E.ABC CALLED FROM P.WXYZ_217B

- ? TRACEBACK,MONTH
  P.MONTH CALLED FROM P.FORMAT_13B
  P.FORMAT CALLED FROM P.DATE_71B
  P.DATE CALLED FROM P.PROGA_152B

LANGUAGE-DEPENDENT COMMANDS

Language-dependent CID commands are nearly identical in form and action to statements used in the programming language of the program being debugged. These commands are provided by CID so that you can debug programs without having to be familiar with compiler-produced data structures and object code.

Language-dependent commands are available only when the home program is a BASIC, COBOL, or FORTRAN program compiled for use with CID. Different commands are available for debugging programs written in different languages. You cannot enter language-dependent commands intended for one programming language when the home program is written in a different programming language.

The syntax of a language-dependent command depends on the command itself. Normally, each line entered from the terminal is considered a complete command. More than one command can be entered on a line if a semicolon separates the commands (two semicolons must be used after the BASIC PRINT command). Language-dependent commands can be entered on the same line with language-independent commands.

Commands cannot be continued across lines. The maximum length of a command line is 150 characters. Characters beyond 150 are ignored. CID can be used from a NOS ASCII mode terminal. The command line is limited to a maximum of 75 12-bit escape code ASCII characters. CID can be used to debug ASCII mode BASIC programs under NOS. NOS/BE, however, does not support input or output of lowercase ASCII characters with CID.

BASIC CID COMMANDS

Five CID commands are provided exclusively for debugging BASIC programs. They are GOTO, IF, LET, MAT PRINT, and PRINT. These commands have the same syntax and function as equivalent BASIC statements except for the following restrictions (and those noted in the command descriptions):

- Arithmetic or string expressions cannot refer to system- or user-defined functions.
- Arithmetic expressions cannot contain the exponentiation operation ^ or **.
- Multiple commands can appear on the same line if they are separated by semicolons; however, two semicolons are required to separate a PRINT or MAT PRINT command from following commands.

Subscript expressions and the substring notation are allowed in BASIC CID commands.

The language-independent DISPLAY, ENTER, MOVE, and SKIPIF commands should not in general be used with BASIC program data; the corresponding BASIC commands should be used instead. ENTER and MOVE must not be used with BASIC string variables.

The main advantage of the BASIC CID commands, apart from their familiarity to BASIC programmers, is that they provide automatic output formatting (PRINT) or referencing by variable name (LET and IF).

Expressions used with these commands are used as a value and not as an address. They follow the syntax and operator precedence rules of BASIC expressions, except that function references and exponentiation are not allowed. Variables and line numbers specified in the BASIC CID commands must be in the home program.

GOTO COMMAND

The GOTO command (figure 6-1) is used to resume program execution at a designated line. Care should be taken when you enter this command, because it changes the flow of program execution. You should not use this command to initiate execution at the beginning of a debug session, because program initialization will not take place.

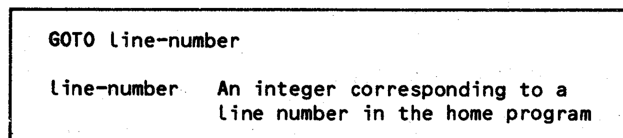


Figure 6-1. GOTO Command

IF COMMAND

The IF command (figure 6-2) is used to control the selection of CID commands based on a comparison of program variables or computed values. The logical operators AND, OR, and NOT are supported. They can be used to connect simple relational expressions. IF THEN ELSE is not supported.

Variables used in the expressions must exist in the program. Expressions are evaluated and compared according to BASIC rules. If the specified relation is true, the CID command is executed. If the relation is false, the CID command is skipped.

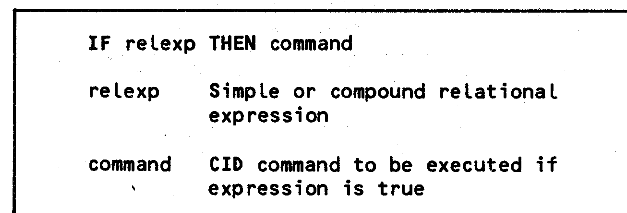


Figure 6-2. IF Command

Examples of the IF command are:

- IF A>=5000 THEN LET A=5000
- IF A$<>B$ THEN GO
- IF A(I,3)=B+C/12.3 THEN PAUSE
- IF A$<=B$(I:3) AND A$="YES" THEN GOTO 300

LET COMMAND

The LET command (figure 6-3) is used to assign a value to a program variable. The variable referenced must exist in the BASIC program being debugged. The expression can be assigned to only one variable. All string and arithmetic operators except exponentiation are supported.

```
[LET] numeric-variable = arithmetic-expression

[LET] string-variable = string-expression
```

Figure 6-3. LET Command

Examples of the LET command are:

- LET A=A+45
- LET Z=A
- LET B$(3,K)=A$+"ABC"

MAT PRINT COMMAND

The MAT PRINT command (figure 6-4) is used to print one-, two-, or three-dimensional arrays.

```
MAT PRINT array-list

array-list   List of one or more arrays
             separated by commas or semicolon
```

Figure 6-4. MAT PRINT Command

Arrays listed in the array list must exist in the program. The subscripts furthest to the right in the dimension statement vary and print more rapidly than the other subscripts in the array list.

Elements of the array are printed in row order with spacing between items controlled by the comma or semicolon as it is for the PRINT command. A blank line is output after each row. For a three-dimensional array, each plane is printed in row order. Two blank lines separate one plane from another. An extra blank line is output between matrices.

The MAT PRINT command is the only mechanism available for printing an entire array.

Two semicolons in succession separate MAT PRINT from the next command on the line.

Assuming the base set in the OPTION statement is one, a 2x2 array named A is printed as follows when the MAT PRINT is executed:

```
A(1,1) A(1,2)
A(2,1) A(2,2)
```

A 2x2x2 array named A is printed as follows:

```
A(1,1,1) A(1,1,2)
A(1,2,1) A(1,2,2)

A(2,1,1) A(2,1,2)
A(2,2,1) A(2,2,2)
```

The OPTION statement is described in the BASIC reference manual.

PRINT COMMAND

The PRINT command (figure 6-5) is used to print values of program variables or expressions. Neither the TAB function nor the capability to print a partial line is supported. Any trailing comma is ignored. List elements can be separated by commas or single semicolons. Two semicolons in succession separate a PRINT command from the next command on the same line.

```
PRINT output-list

output-list   Any number of arithmetic or
              string expressions separated by
              commas or semicolons
```

Figure 6-5. PRINT Command

Variables in the output list must exist in the program. A comma separating expressions causes spacing to the next 15-column print zone; a semicolon causes no spacing.

Examples of the PRINT command are:

- PRINT "THE VALUE OF A IS"; A
- PRINT A,A*A,A*A*A+13,J,3+B(17,J)

COBOL CID COMMANDS

Four special commands are provided exclusively for debugging COBOL programs. The four special COBOL CID commands are DISPLAY, GO TO, MOVE, and SET. The COBOL CID commands are similar in syntax and action to the standard COBOL statements having the same names. A COBOL CID command can be terminated by a period; however, the period is unnecessary. A semicolon must separate commands entered on the same line.

The COBOL CID commands described in this section have the same names as the language-independent DISPLAY, GO, MOVE, and SET commands described in section 5. However, in action, the commands produce completely different results. To avoid ambiguities when the home program is a COBOL program compiled for use with CID, you should enter

language-independent commands (described in section 5) in short form or with a comma following the name of the command. CID assumes any ambiguous command is a COBOL CID command.

The COBOL CID commands allow you to examine and change COBOL data items in a way identical to writing such commands inside a COBOL program. Data items and procedures are referenced just as within a COBOL program. The COBOL CID commands described in this section largely preclude the need for using the language-independent DISPLAY, ENTER, and MOVE commands described in section 5; however, the following restrictions apply to the COBOL CID commands described in this section:

- Only data items declared in the home program can be referenced in COBOL CID commands.
- Debug variables cannot be referenced by the COBOL CID commands.

You can use COBOL CID commands to reference data items outside the current home program by changing the designation of the home program. (The SET,HOME command changes home program designation.) Data items from more than one program cannot be output by a single COBOL CID DISPLAY command, nor can data be moved from one module to another with a COBOL CID MOVE or SET command.

Other considerations regarding COBOL CID commands are as follows:

- No computations, other than expressions in reference modifiers, are permitted with COBOL data items in the COBOL CID commands.
- Reference modification is permitted in COBOL CID commands. Expressions and identifiers are permitted in reference modifiers in COBOL CID commands.
- Use of indexes is permitted in COBOL CID commands; however, CID does not detect use of an index name outside of the data items for which it was defined.
- Only quotation marks (") can be used to delimit nonnumeric literals, even if the source program contains the QUOTE IS APOSTROPHE clause in the SPECIAL-NAMES paragraph.
- COBOL source statement column and area rules do not apply to CID commands. COBOL CID commands can be entered at the beginning of the line.

The following paragraphs describe the COBOL CID commands.

DISPLAY COMMAND

The DISPLAY command (figure 6-6) displays the values of literals and data items. The values are displayed in the same format produced by a COBOL DISPLAY statement. The UPON and WITH NO ADVANCING phrases must not be used in the DISPLAY command.

```
DISPLAY output-list

output-list   List of literals and identifiers
              to be displayed
```

Figure 6-6. DISPLAY Command

The ANSI=NOEDIT and ANSI=AUDIT options in the COBOL5 control statement do not affect the output caused by the COBOL CID DISPLAY command. Numeric data items are always edited.

If the DECIMAL POINT IS COMMA clause is given in the SPECIAL NAMES paragraph of the COBOL source program, output from the DISPLAY command uses a comma for the decimal point. Otherwise, a period is output as the decimal point.

Examples of the DISPLAY command are:

- DISPLAY I, AMOUNT-PAID, ID-CODE(2:5)
- DISPLAY "**** TOTAL VALUE **** ", TOTAL
- DISPLAY TOTALS OF SUMS OF REPORTS

GO TO COMMAND

The GO TO command (figure 6-7) resumes program execution at the beginning of a paragraph or section. The DEPENDING ON phrase must not be used with the GOTO command.

```
GO [TO] place

place   Name of COBOL procedure where execution
        is to resume
```

Figure 6-7. GO TO Command

The GO TO command must not be used to initiate program execution at the beginning of a debug session because program initialization will not take place. The language-independent GO command, the EXECUTE command, or the STEP command should be specified to initiate execution. These commands are described in section 5. Care should also be taken when you enter the GO TO command, because it changes the flow of program execution.

Examples of the GO TO command are:

- GO TO CALC-RANGE

  Causes execution to resume at the procedure CALC-RANGE in the home program.

- GO CALC-COURSE OF DEVELOP-ALTERNATE

  Causes execution to resume at the CALC-COURSE paragraph in the DEVELOP-ALTERNATE section in the home program.

MOVE COMMAND

The MOVE command (figure 6-8) changes the value of a data item. This command is similar to the COBOL MOVE statement except as follows:

- Allowable types of sending and receiving items are restricted.
- Only one receiving item is permitted.
- MOVE CORRESPONDING is not allowed.

```
MOVE value TO data-item

value        Literal or identifier

data-item    Identifier of data item to receive
             value
```

Figure 6-8. MOVE Command

Restrictions on the types of sending and receiving items in the MOVE command are as follows:

- The sending and receiving items must be alphabetic, alphanumeric, numeric (other than COMP-1 or COMP-4), or group items. Edited items can be specified; however, a warning message is issued, and no editing takes place when the value is moved.
- If either the sending or receiving item is COMP-2, both items must be COMP-2.
- The receiving item must not be a Report Section item.
- The sending item must not be a figurative constant.

Allowable sending and receiving items are shown in table 6-1.

Examples of the MOVE command are:

- MOVE 57 TO SCORE(1)

  Gives the value 57 to the first member of the table SCORE.

- MOVE "B" TO GRADE OF STUDENT-A

  Gives the value "B" to data item GRADE in the group item STUDENT-A.

- MOVE PERCENTAGE TO SCORE (1)

  Gives the value of PERCENTAGE to the first member of the table SCORE.

SET COMMAND

The SET command sets the value of an index or data item. The SET command has two formats.

Format 1 SET Command

The format 1 SET command (figure 6-9) sets an index or data item to a specified value.

Not all combinations of name types and value types are allowed in the format 1 SET command. Allowable combinations of name types and value types are indicated in table 6-2.

```
SET name TO value

name     Index name or identifier to be set to
         value

value    Integer, identifier, or index name
```

Figure 6-9. Format 1 SET Command

TABLE 6-1. ALLOWABLE MOVE COMMAND SENDING AND RECEIVING ITEMS

Sending Item	Receiving Item				
	Group	Alphabetic	Alphanumeric	Numeric	COMP-2
Group	Yes†	Yes	Yes	Yes	No
Alphabetic	Yes†	Yes	Yes	No	No
Alphanumeric	Yes†	Yes	Yes	Yes	No
Numeric	Yes†	No	Yes	Yes	No
COMP-2	No	No	No	No	Yes

†CID issues a warning message before this kind of move takes place.

TABLE 6-2. ALLOWED FORMAT 1 SET COMMAND OPERATIONS

Value	Name		
	Integer Data Item	Index Name	Index Data Item
Integer literal	No	Yes	No
Integer data item	No	Yes	No
Index name	Yes	Yes	Yes
Index data item	No	Yes	Yes

An example of the format 1 SET command is:

```
SET INDEX-A TO 13
```

This command sets the index INDEX-A to the value 13.

Format 2 SET Command

The format 2 SET command (figure 6-10) increments or decrements the value of the index specified.

```
SET name UP BY amount
SET name DOWN BY amount

name     Index name to be set up or down by the
         amount

amount   Integer or identifier of an elementary
         numeric data item
```

Figure 6-10. Format 2 SET Command

The UP BY clause causes the SET command to add the amount to the named index; the DOWN BY clause causes the SET command to subtract the amount from the named index.

Examples of the format 2 SET command are:

- SET INDEX-A DOWN BY 1

  Subtracts 1 from INDEX-A.

- SET INDEX-B UP BY 5

  Adds 5 to INDEX-B.

FORTRAN CID COMMANDS

Four special CID commands are provided exclusively for debugging FORTRAN programs. They are assignment, GOTO, IF, and PRINT. They are similar in syntax and action to the standard FORTRAN statements having the same names. These commands allow you to examine, change, and test FORTRAN variables in a way identical to writing such commands inside a FORTRAN program itself. Subscripted variables and substrings are referenced just as within a FORTRAN program. The FORTRAN CID commands largely preclude the need for using DISPLAY, ENTER, MOVE, and SKIPIF commands except for the following restrictions:

- FORTRAN CID commands do not allow qualification of variables; only variables declared in the home program can be referenced.

- FORTRAN CID commands cannot reference debug variables.

The SET,HOME command can be used to change the home program as required to overcome the first restriction. However, variables from more than one program cannot be output by a single PRINT statement, nor can data be moved from one module to another with an assignment statement. References to common block variables declared in the home program module can, however, appear in the same statement with home program variables. The scope of FORTRAN variables under CID is identical to that in FORTRAN; they are confined to the subprograms in which they are declared.

The main advantage of these commands, apart from their familiarity to FORTRAN programmers, is that they provide automatic output formatting (PRINT) or automatic type conversion (assignment and IF) of the referenced variables.

Expressions used with these commands are interpreted as values and not as addresses. They follow the syntax and operator precedence rules of FORTRAN expressions, except that function references and exponentiation are not allowed.

ASSIGNMENT COMMAND

Assignment commands (figure 6-11) are identical in form and action to FORTRAN assignment statements. Any valid FORTRAN expression is supported, except those involving function references or exponentiation. Only one variable can change its value as a result of the assignment command.

```
simple variable = FORTRAN expression
subscripted variable = FORTRAN expression
```

Figure 6-11. Assignment Command

Examples of the assignment command are:

- A=B

- VECT(3)=ARRAY(3,7)

- VOL=(4/3)*3.14159*R*R*R

- NAME=FIRST//LAST (FORTRAN 5 only)

- LINE(3:9)=BLANKS(1:6) (FORTRAN 5 only)

GOTO COMMAND

The GOTO command (figure 6-12) resumes program execution at the specified statement. Care should be taken when you enter this command, because it changes the flow of program execution. You should not use this command to initiate program execution at the beginning of a debug session, because program initialization will not take place.

```
GOTO statement-label

statement-label   Integer corresponding to
                  statement number in home
                  program of statement where
                  execution is to begin.  The
                  location must be in the home
                  program.
```

Figure 6-12. GOTO Command

IF COMMAND

The IF command (figure 6-13) is identical in form and action to the FORTRAN logical IF statement.

```
IF (logical expression) command
```

Figure 6-13. IF Command

The IF command executes the consequent command if the logical expression is true. The logical expression can be any FORTRAN logical expression not involving function references or exponentiation. The consequent CID command can be any command.

Examples of the IF command are:

- IF (X .EQ. Y) PRINT *,GLOP+GLIP
- IF (STAT(1:2) .NE. 'OK') PAUSE

NOTE

Character relational expressions are evaluated according to the current collation weight table. The FORTRAN 5 CS=FIXED collation option is never used even if the program being debugged is currently using that option.

PRINT COMMAND

The PRINT command (figure 6-14) is identical in form and action to the FORTRAN list-directed PRINT statement; output from this command is sent to the CID standard output file. The PRINT command formats each list element according to its type.

```
PRINT *,output-list

List elements are separated by commas and can
include any of the following:

●  Simple variable

●  Subscripted variable

●  Constant

●  Expression not involving exponentiation or
   functions

●  Implied DO list enclosed in parentheses
   (same as in FORTRAN input/output list)

●  Substrings (FORTRAN 5 only)

●  Array name

●  Character variables and expressions
   (FORTRAN 5 only)
```

Figure 6-14. PRINT Command

Examples of the PRINT command are:

- PRINT *,I,B(2,7),TOTAL/UNITS
- PRINT *,((MATRIX(I,J),I=1,3),J=1,4)
- PRINT *,(VECTOR(I),I=1,3)
- PRINT *,A(2:5),CHAR//LINE (FORTRAN 5 only)
- PRINT *,NAME

As in FORTRAN itself, there is no conflict between an implied DO variable and a variable in the home program having the same name.

An array declared with an upper bound of asterisk must be referenced as a subscripted variable in the print command. A character variable declared with a length of asterisk must be referenced with substring notation.

ERROR, WARNING, VETO MODE, AND INTERRUPT PROCESSING

7

This section describes four conditions in which execution of CID commands is suspended and control is given to you to respond to the encountered condition. The conditions are errors, warnings, veto mode, and interrupts. Possible responses to the conditions are described.

This section describes processing of the conditions when you are using CID interactively. For batch processing differences, see appendix E.

ERROR PROCESSING

If any CID command cannot be successfully executed, an error message is issued and the system is placed in error response mode. In this temporary interactive mode, the system awaits an appropriate response before dealing with the error situation.

If the command in error was issued in interactive command mode, then the error message consists simply of *ERROR-, followed by the error text on the same line.

An example of the occurrence of an error is:

```
? ENTER 100000 *FL
*ERROR-INVALID PARAMETER TYPE *FL
?
```

If the error occurs in a command sequence (such as a breakpoint, trap, or group body) or in a group of commands on one line other than as the first command, the command in error is displayed. The display includes any delimiter following the command. This character, if present, should be checked for a left or right bracket, indicating, respectively, the start or end of a collected sequence.

An example of an error occurring in a line sequence is:

```
? DISPLAY,#P;ENTER,100,#FL
  #P=P.SUB1_173B
  *CMD - (ENTER,100,#FL)
     *ERROR-INVALID PARAMETER TYPE #FL
?
```

A complete list of error messages can be found in appendix B. Responses you can make to error messages are described later in this section.

WARNING PROCESSING

If a CID command might have unexpected results, a warning is issued. For example, attempted execution of the SET,BREAKPOINT command results in a warning message if a breakpoint already exists at the location given, or if the location is an entry point but was not given as such.

After issuing the warning message, the system is placed in warning response mode, a temporary interactive mode. Processing is suspended until you respond to the message.

Warnings are reported in the same manner as errors, except that the message text begins with *WARN- instead of *ERROR-, and CID prompts you with OK? rather than just ?.

A complete list of warning messages can be found in appendix B. Responses you can make to warnings are described later in this section.

VETO MODE PROCESSING

Veto mode provides a method of command sequence operation that combines the automatic and interactive modes. When veto mode is on, CID displays each command in a command sequence immediately prior to execution of the command.

CID then gives you temporary control, and issues the prompt OK?. You can choose to allow the current command to be executed, skipped (vetoed), or replaced by one or more supplied commands. Other options allow the current line or remainder of the current sequence to be omitted, or allow veto mode to be inhibited for the remainder of the current line or current command sequence.

On the same response line, you can also supply additional commands to be executed prior to resumption of the command sequence. These additional commands are not subject to veto. Responses you can make to the OK? prompt in veto mode are described later in this section.

The SET,VETO and CLEAR,VETO commands described in section 5 turn veto mode on and off. Veto mode is off until turned on by the SET,VETO command.

INTERRUPT PROCESSING

You can obtain interactive CID control by issuing a terminal interrupt (see interrupt in the glossary, appendix C, for a description of how to issue a terminal interrupt). If your program is executing when the issued interrupt is detected, an INTERRUPT trap occurs, as described in section 3. However, if a command sequence is executing when the interrupt is detected, then a halt in execution of the command sequence occurs. Normally the halt occurs between CID commands. However, for some commands that take lists as parameters, interruption of the command occurs after the current list element is processed; for commands that print more than one line of output, interruption occurs after the current line is printed.

An interrupt can also be detected when CID is in collect mode. CID can be in collect mode when you are creating a command sequence or when an executing command sequence is creating another command sequence (a command sequence can create another command sequence, for example, by setting a trap with a trap body). Collect mode is described in section 3.

If CID is in collect mode when the terminal interrupt is detected, the following message is issued:

    *IN COLLECT MODE, LEVEL n

This message informs you that CID is in collect mode, where n is an integer indicating the number of levels of collect mode; that is, the number of right brackets that must be entered to terminate collect mode. (If n=1, the level number is omitted from the message.)

If the terminal interrupt is detected while you already have control, the interrupt is ignored and the following informative message is issued:

    *INTERRUPT IGNORED

Responses you can make to interrupts are described later in this section.

ERROR, WARNING, VETO MODE, AND INTERRUPT RESPONSES

Whenever you are prompted for a response as the result of an error, warning, veto, or interrupt, CID is put in response mode and you can respond in one of three ways:

- Format 1:

  Command Line

  You can enter any command or set of commands separated by semicolons. These commands are executed in place of the command that caused the error, warning, or veto; in the case of an interrupt, they are additional commands executed at the point in the original command sequence where the interrupt was recognized. Unless one of the replacement commands causes control to go elsewhere (for example, GO, EXECUTE, JUMP, or SKIPIF), control returns to the next command in the original sequence after all commands in the response are processed. In response to an error, warning, or veto, the NULL command (NULL or just semicolon) is a valid reply that effectively skips the command in question.

  If you enter a PAUSE command in response to an error, warning, interrupt, or veto, the command sequence is suspended, and you can enter several command lines from the terminal. You can resume execution of the sequence by entering the GO command or resume execution of the program by entering the EXECUTE command.

- Format 2:

  Response keyword[,qualifier]

  The response keyword is a positive keyword (ACCEPT, YES, or OK) or a negative keyword (REJECT, NO, or VETO); the qualifier, if present, is LINE or SEQ. Each positive keyword has the same effect as any other positive keyword, and each negative keyword has the same effect as any other negative keyword. The effects of the response keywords and their qualifiers are shown in table 7-1.

- Format 3:

  Response keyword[,qualifier];command line

  This combination response responds to the condition as specified by the keyword and then inserts the new command line at the current position in the original sequence. It provides the capability to accept a warning or veto and insert some new commands as well.

  Qualified negative keywords such as REJECT,LINE and VETO,SEQ can be used in these combination results; they are not very useful, however, since the new commands are treated as if they existed in the original line and are consequently skipped.

If a response line containing multiple commands is interrupted, or if any of the commands in the response line causes an error or warning prompt, the remaining commands in the response line are executed after processing of the inserted response sequence. However, if the total number of pending command sequences is 16, the last response sequence is discarded; a message to this effect is issued.

Commands entered in a response line are not subject to veto.

A response keyword, if present at all, must be first on the response line; otherwise, it is diagnosed as an illegal command. If a response keyword is encountered in other than a response input line, it is diagnosed as an illegal command.

ERROR RESPONSES

Examples of error responses are shown in table 7-2.

WARNING RESPONSES

Examples of warning responses are shown in table 7-3.

VETO MODE RESPONSES

Examples of response lines while in veto mode are shown in table 7-4.

INTERRUPT RESPONSES

The most useful responses to an interrupt occurring when not in collect mode are shown in table 7-5.

TABLE 7-1. RESPONSE KEYWORD ACTIONS

Response	Condition			
	Error (?)	Warning (OK?)	Veto (OK?)	Interrupt (?)
Pos (ACCEPT, OK, YES)	Acknowledge the error and return to the original sequence; that is, skip the bad command (same as responding with NULL command).	Execute the command.	Execute the command.	Acknowledge interrupt and return to original sequence (same as responding with NULL command).
Pos, LINE	Same as Pos. Not possible to inhibit error processing for rest of line.	Same as Pos.	Execute current command and all other commands on the current line; that is, inhibit veto until a new line of commands is encountered.	Same as Pos.
Pos, SEQ	Same as Pos. Not possible to inhibit error processing for the rest of the current sequence.	Same as Pos.	Execute current command and inhibit veto for the remainder of this sequence. Reinstate veto when the current sequence is complete.	Same as Pos.
Neg (REJECT, VETO, NO)	Same as Pos. The bad command is rejected.	Do not execute the current command. Discard it and return to the original sequence to process next command (same as responding with NULL command).	Do not execute current command. Discard it and return to the original sequence for next command (same as responding with NULL command).	Same as Pos.
Neg, LINE	Reject current command and all others on the line it came from; that is, skip to the next line of commands. (This effect cannot be simulated by a replacement command.)	Same as for error case.	Same as for error case. Veto is still in effect when next line of commands is processed.	Same as for error case.
Neg, SEQ	Reject current command and all others in this sequence (same as responding with GO command).	Same as for error case.	Same as for error case. Veto is still in effect when commands in sequence that invoked this one are processed.	Same as for error case.

TABLE 7-2. ERROR RESPONSE EXAMPLES

Response	Action
OK	Skip the current command.
NO	Skip the current command.
NULL	Skip the current command.
;	Skip the current command.
HELP,SYNTAX;PAUSE	Replace the command with HELP,SYNTAX, then enter interactive input mode. If GO is issued later, the command sequence is resumed.
REJECT,LINE	Reject the current command and all others on the same line that are still unprocessed.
VETO,SEQ	Terminate the current command sequence.
GO	Same as VETO,SEQ.

TABLE 7-3. WARNING RESPONSE EXAMPLES

Response	Action
OK	Execute the current command.
NO	Reject the current command.
NULL or ;	Reject the current command.
REJECT,LINE	Reject the current command and all others on the same line that are still unprocessed.
REJECT,SEQ	Reject the current command and the rest of this sequence.
GO	Same as REJECT,SEQ.

TABLE 7-4. VETO MODE RESPONSE EXAMPLES

Response	Action
OK	Execute the current command.
NO	Skip the current command.
NULL	Replace current command with NULL. Action is identical to NO.
;	Action is identical to NO.
OK;D #P	Execute the current command and then display #P. Note D #P is not subject to veto.
YES,SEQ;D #V1	Execute the current command, display #V1, then inhibit veto for the rest of the command sequence.

TABLE 7-5. INTERRUPT RESPONSE EXAMPLES

Response	Action
PAUSE	Enter interactive mode. If GO is later issued, the command sequence is resumed.
OK;PAUSE	Same as PAUSE.

If an interrupt is detected while CID is in collect mode, and the interrupt is not ignored (that is, if the interrupt occurs while an executing command sequence is creating another command sequence), you can choose one of the following actions:

- Resume collection of the command sequence.

  You can immediately resume collection of the current command sequence only by entering one of the following responses:

  ACCEPT

  YES

  OK

  NULL

  ;

  These responses cause collection of the command sequence to resume. You can then issue another terminal interrupt and hope that this time the interrupt does not occur while in collect mode.

- Terminate collection of the command sequence.

  You can terminate collection of the command sequence by entering as many right brackets (]) as necessary to get out of collect mode, followed by the PAUSE command. When collection is terminated in this manner, the interrupted command sequence remains suspended at the point where the interrupt was detected. Therefore, if you enter the GO command, the commands that would have been collected if the interrupt had not occurred are executed instead. For example, assume the command sequence in figure 7-1 is executing, and an interrupt is detected just before the command DISPLAY,#LINE is collected. When the interrupt is detected, CID issues the message:

  *IN COLLECT MODE

  If you enter a right bracket and then the GO command, the command DISPLAY,#LINE is executed, rather than collected. When the right bracket is encountered in the command sequence, CID issues an error message.

```
    .
    .
    .
DISPLAY,#V8
SET,TRAP,LINE,* [
DISPLAY,#LINE
]
ENTER,#V8+1,#V8
    .
    .
    .
```

Figure 7-1. Command Sequence For Interrupt Example

You should not enter CID commands (other than the NULL command) before terminating or resuming collection of the command sequence; any commands entered would be collected, and CID does not indicate to you in which command sequence (or where in that sequence) the commands are being collected.

SAMPLE DEBUG SESSIONS 8

Several debug sessions are illustrated. Session A (figures 8-1, 8-2, and 8-3) shows a single module COMPASS program that prints the date and time. Session B (figures 8-4 and 8-5) shows a FORTRAN program consisting of a main program and several subroutines. The detection of an undefined variable is illustrated. Session C (figures 8-6, 8-7, and 8-8) shows a FORTRAN program that contains overlays. Session D (figures 8-9, 8-10, 8-11, and 8-12) shows a COBOL main program with a FORTRAN subroutine. In the sample sessions, user input is indicated by lowercase letters; CID output is uppercase.

```
                 IDENT     DAYTIME
                 ENTRY     START
START            BSS       0
                 DATE      A                  GET DATE IN A    YY/MM/DD.
                 CLOCK     B                  GET TIME IN B    HH.MM.SS.
                 SB1       1
                 SA1       A                  MOVE DATE TO BUFFER
                 BX6       X1
                 SA6       BUF
                 SA2       BLANKS             PUT 10 BLANKS IN BUFFER
                 BX7       X2
                 SA7       A6+B1
                 SA1       B                  MOVE TIME TO BUFFER
                 BX6       X1
                 SA6       A7+B1
                 MX6       0                  END OF LINE TO BUFFER
                 SA6       A6+B1
                 SX7       A6+B1              SET OUTPUT IN POINTER
                 SA7       OUTPUT+2
                 WRITER    OUTPUT,RECALL      WRITE BUFFER OUT
                 ENDRUN                       FINISH
A                BSS       1
B                BSS       1
BLANKS           DATA      10H
BUF              BSS       65
OUTPUT           FILEB     BUF,65
                 END       START
```

Figure 8-1. Session A Source Listing

```
/debug(on)
$DEBUG(ON)
/lgo
 CYBER INTERACTIVE DEBUG
? set auxiliary auxfl ewidrbt
? list, map
 DBUG.,    DAYTIME,    CPU.CIO,    CPU.SYS,    UCLOAD
? lm p.daytime
 PROGRAM - DAYTIME,    FWA = 3247B,    LENGTH = 130B
 ENTRY POINTS - START
? set trap, store p.daytime
 INTERPRET MODE TURNED ON
? set trap, fetch p.daytime
? go
 *T #2, FETCH OF _17B AT _7B
? d #ew c
 #EW =  80/10/20.
? go
 *T #1, STORE INTO _22B IN _7B
? d #ew c
 #EW =  80/10/20.
```

Figure 8-2. Debug Session A (Sheet 1 of 2)

```
? go
 *T #2, FETCH OF _21B IN _10B
? d #ew c
 #EW =
? go
 *T #1, STORE INTO _23B AT _11B
? d #ew c
 #EW =
? go
 *T #2, FETCH OF _20B IN _11B
? d #ew c
 #EW =  10.32.10.
? go
 *T #1, STORE INTO _24B AT _12B
? d #reg
 A0 = 004000    X0 = 00000 00000 00000 00000    B0 = 000000
 A1 = 003267    X1 = 55343 35736 35573 43357    B1 = 000001
 A2 = 003270    X2 = 55555 55555 55555 55555    B2 = 000002
 A3 = 000057    X3 = 00000 00000 00000 00000    B3 = 015001
 A4 = 000001    X4 = 00000 00000 00000 00000    B4 = 004001
 A5 = 000317    X5 = 60000 00000 04004 00000    B5 = 000317
 A6 = 003273    X6 = 55343 35736 35573 43357    B6 = 004000
 A7 = 003272    X7 = 55555 55555 55555 55555    B7 = 037756
? d #x c
 X0 =
::::::    X1 =  10.32.10.
 X2 =                X3 =

 X4 =
::::::    X5 = #
D:5
 X6 =  10.32.10.    X7 =
? go
 *T #1, STORE INTO _25B IN _12B
? d _21b,,4
 _21B = 55555 55555 55555 55555    55433 35034 33503 53357
      " +2 = 55555 55555 55555 55555    55343 35736 35573 43357
? d _21b c 4
 _21B =             80/10/20.            10.32.10.
? go
 *T #1, STORE INTO _125B IN _13B
? go
 *T #2, FETCH OF P.DAYTIME_123B IN P.CPU.CIO_7B
? go
 *T #2, FETCH OF P.DAYTIME_123B IN _2B
? go
 *T #1, STORE INTO P.DAYTIME_123B IN _4B
? go
 80/10/20.            10.32.10.
 *T #17, END IN P.CPU.SYS_4B
? d #reg
 A0 = 004000    X0 = 00000 00000 00000 00000    B0 = 000000
 A1 = 000001    X1 = 00000 00000 00000 00000    B1 = 000001
 A2 = 003270    X2 = 00000 00000 00000 03372    B2 = 000002
 A3 = 000057    X3 = 00000 00000 00000 00000    B3 = 015001
 A4 = 000001    X4 = 00000 00000 00000 00000    B4 = 004001
 A5 = 000317    X5 = 60000 00000 04004 00000    B5 = 000317
 A6 = 000001    X6 = 05160 42000 00000 00000    B6 = 004000
 A7 = 003372    X7 = 00000 00000 00000 00000    B7 = 037756
? list status
 HOME = P.CPU.SYS,   NO BREAKPOINTS,   2 TRAPS,   NO GROUPS,   VETO OFF
 INTERPRET ON,   OUT OPTIONS = I W E D
 AUX FILE = AUXFL, OPTIONS = I W E D R B T
? quit
 DEBUG TERMINATED
```

Figure 8-2. Debug Session A (Sheet 2 of 2)

```
LIST, MAP
DBUG.,   DAYTIME,   CPU.CIO,   CPU.SYS,   UCLOAD
LM P.DAYTIME
PROGRAM - DAYTIME,   FWA = 3247B,   LENGTH = 130B
ENTRY POINTS - START
SET TRAP, STORE P.DAYTIME
INTERPRET MODE TURNED ON
SET TRAP, FETCH P.DAYTIME
GO
*T #2, FETCH OF _17B AT _7B
D #EW C
#EW =  80/10/20.
GO
*T #1, STORE INTO _22B IN _7B
D #EW C
#EW =  80/10/20.
GO
*T #2, FETCH OF _21B IN _10B
D #EW C
#EW =
GO
*T #1, STORE INTO _23B AT _11B
D #EW C
#EW =
GO
*T #2, FETCH OF _20B IN _11B
D #EW C
#EW =  10.32.10.
GO
*T #1, STORE INTO _24B AT _12B
D #REG
A0 = 004000    X0 = 00000 00000 00000 00000     B0 = 000000
A1 = 003267    X1 = 55343 35736 35573 43357     B1 = 000001
A2 = 003270    X2 = 55555 55555 55555 55555     B2 = 000002
A3 = 000057    X3 = 00000 00000 00000 00000     B3 = 015001
A4 = 000001    X4 = 00000 00000 00000 00000     B4 = 004001
A5 = 000317    X5 = 60000 00000 04004 00000     B5 = 000317
A6 = 003273    X6 = 55343 35736 35573 43357     B6 = 004000
A7 = 003272    X7 = 55555 55555 55555 55555     B7 = 037756
D #X C
X0 =
::::::    X1 =  10.32.10.
X2 =                X3 =
X4 =
::::::    X5 = #
D:5
X6 =  10.32.10.    X7 =
GO
*T #1, STORE INTO _25B IN _12B
D _21B,,4
_21B = 55555 55555 55555 55555     55433 35034 33503 53357
    " +2 = 55555 55555 55555 55555     55343 35736 35573 43357
D _21B C 4
_21B =              80/10/20.             10.32.10.
GO
*T #1, STORE INTO _125B IN _13B
GO
*T #2, FETCH OF P.DAYTIME_123B IN P.CPU.CIO_7B
GO
*T #2, FETCH OF P.DAYTIME_123B IN _2B
GO
*T #1, STORE INTO P.DAYTIME_123B IN _4B
GO
*T #17, END IN P.CPU.SYS_4B
```

Figure 8-3. Session A Auxiliary File Listing (Sheet 1 of 2)

```
D #REG
A0 = 004000    X0 = 00000 00000 00000 00000    B0 = 000000
A1 = 000001    X1 = 00000 00000 00000 00000    B1 = 000001
A2 = 003270    X2 = 00000 00000 00000 03372    B2 = 000002
A3 = 000057    X3 = 00000 00000 00000 00000    B3 = 015001
A4 = 000001    X4 = 00000 00000 00000 00000    B4 = 004001
A5 = 000317    X5 = 60000 00000 04004 00000    B5 = 000317
A6 = 000001    X6 = 05160 42000 00000 00000    B6 = 004000
A7 = 003372    X7 = 00000 00000 00000 00000    B7 = 037756
LIST STATUS
HOME = P.CPU.SYS,   NO BREAKPOINTS,   2 TRAPS,   NO GROUPS,   VETO OFF
INTERPRET ON,   OUT OPTIONS = I W E D
AUX FILE = AUXFL, OPTIONS = I W E D R B T
QUIT
```

Figure 8-3. Session A Auxiliary File Listing (Sheet 2 of 2)

```
1 FTN 5.1+538          81/07/15. 14.13.20 PAGE      1
     PROGRAM FACTORS  74/74    OPT=0

         1          PROGRAM FACTORS
         2 C       THIS MAIN PROGRAM CALLS SUBROUTINES THAT READ A LIST OF
         3 C       INTEGERS AND (INTEGER) FIRST FACTORS, SORT THE NUMBERS,
         4 C       AND FIND THE SECOND INTEGER FACTORS.  IF THE FIRST FACTOR
         5 C       IS NOT A TRUE FACTOR, "NON-INTEGER" IS PRINTED OUT
         6 C       INSTEAD OF THE SECOND FACTOR.
         7 C
         8          DIMENSION LIST(3,10)
         9          CALL GETLIST(LIST)
        10          CALL OUTLIST(N,LIST)
        11          END
1 FTN 5.1+538          81/07/15. 14.13.20 PAGE      1
  SUBROUTINE GETLIST  74/74    OPT=0

         1          SUBROUTINE GETLIST (LIST)
         2 C       THIS SUBROUTINE CALLS SUBROUTINES THAT READ THE LIST
         3 C       OF INTEGERS AND FIRST FACTORS, SORT THE NUMBERS,
         4 C       AND FIND THE SECOND FACTORS.
         5 C
         6          DIMENSION LIST (3,10)
         7          CALL GETDATA(N,LIST)
         8          CALL SORTLST(N,LIST)
         9          CALL GETFACT(N,LIST)
        10          RETURN
        11          END
1 FTN 5.1+538          81/07/15. 14.13.20 PAGE      1
  SUBROUTINE GETDATA  74/74    OPT=0

         1          SUBROUTINE GETDATA(N,LIST)
         2 C       THIS SUBROUTINE READS IN THE LIST OF NUMBERS
         3 C       AND FIRST FACTORS.
         4 C
         5          DIMENSION LIST (3,10)
         6          PRINT *, 'TYPE LIST OF NUMBERS AND FIRST FACTORS:'
         7          I=1
         8 100      READ (*,*, END = 200) LIST(1,I), LIST(2,I)
         9          I=I+1
        10          GO TO 100
        11 200      N=I-1
        12          RETURN
        13          END
```

Figure 8-4. Session B FORTRAN Main Program and Subroutines (Sheet 1 of 2)

```
1 FTN 5.1+538           81/07/15. 14.13.20 PAGE     1
  SUBROUTINE SORTLST  74/74    OPT=0

        1          SUBROUTINE SORTLST(N,LIST)
        2 C       THIS SUBROUTINE SORTS THE NUMBERS AND FIRST
        3 C       FACTORS (BY NUMBER).
        4 C
        5          DIMENSION LIST(3,N),NTEMP(2)
        6          DO 1000 I=2,N
        7          DO 500 J=1,I-1
        8          IF (LIST(1,I) .LT. LIST(1,J)) GO TO 700
        9 500      CONTINUE
       10          GO TO 1000
       11 700      NTEMP(1)=LIST(1,I)
       12          NTEMP(2)=LIST(2,I)
       13          DO 800 K=I,J+1,-1
       14          LIST(1,K)=LIST(1,K-1)
       15          LIST(2,K)=LIST(2,K-1)
       16 800      CONTINUE
       17          LIST(1,J)=NTEMP(1)
       18          LIST(2,J)=NTEMP(2)
       19 1000     CONTINUE
       20          RETURN
       21          END
1 FTN 5.1+538           81/07/15. 14.13.20 PAGE     1
  SUBROUTINE GETFACT  74/74    OPT=0

        1          SUBROUTINE GETFACT(N,LIST)
        2 C       THIS SUBROUTINE FINDS THE SECOND FACTORS
        3 C       (GIVES ZERO AS THE SECOND FACTOR IF THE
        4 C       FIRST FACTOR IS NOT A TRUE FACTOR).
        5 C
        6          DIMENSION LIST(3,N)
        7          DO 100 I=1,N
        8          LIST(3,I)=LIST(1,I)/LIST(2,I)
        9          IF (LIST(3,I) * LIST(2,I) .NE. LIST(1,I)) LIST (3,I)=0
       10 100      CONTINUE
       11          RETURN
       12          END
1 FTN 5.1+538           81/07/15. 14.13.20 PAGE     1
  SUBROUTINE OUTLIST  74/74    OPT=0

        1          SUBROUTINE OUTLIST (N,LIST)
        2 C       THIS SUBROUTINE PRINTS OUT THE LIST OF NUMBERS
        3 C       AND FIRST AND SECOND FACTORS. "NON-INTEGER"
        4 C       IS PRINTED OUT AS THE SECOND FACTOR IF THE
        5 C       FIRST FACTOR IS NOT A TRUE FACTOR.
        6 C
        7          DIMENSION LIST(3,N)
        8          PRINT *, '          NUMBER  FIRST   SECOND'
        9          PRINT *, '                  FACTOR  FACTOR'
       10          PRINT *
       11          DO 100 I=1,N
       12          IF (LIST(3,I) .EQ. 0) THEN
       13                 WRITE (*,50) LIST(1,I),LIST(2,I)
       14          ELSE
       15                 WRITE (*,60) LIST(1,I),LIST(2,I),LIST(3,I)
       16          ENDIF
       17 50       FORMAT (5X,2I8,"  NON-INTEGER")
       18 60       FORMAT (5X,3I8)
       19 100      CONTINUE
       20          RETURN
       21          END
```

Figure 8-4. Session B FORTRAN Main Program and Subroutines (Sheet 2 of 2)

```
/ftn5,i=factors,lo=0,db=id
     0.153 CP SECONDS COMPILATION TIME.
/debug,on
DEBUG,ON.
/lgo
 CYBER INTERACTIVE DEBUG
? display,#home
 #HOME = P.FACTORS
? step <------------------------------------------ Initial default STEP command is
 *S LINE AT L.9                                    STEP,1,LINE,*.
? set,trap,line,p.factors
? set,trap,rj,*
 INTERPRET MODE TURNED ON <----------------------- RJ trap turns on interpret mode.
? go
 *T #2, RJ IN L.9
? go
 *T #2, RJ IN P.GETLIST_L.7
? go
 *T #2, RJ IN P.GETDATA_L.6
? set,interpret,off <----------------------------- Turning off interpret mode makes the
? traceback                                        RJ trap ineffective.
 P.GETDATA CALLED FROM P.GETLIST_L.7
 P.GETLIST CALLED FROM P.FACTORS_L.9
? set,breakpoint,l.12
? go
 TYPE LIST OF NUMBERS AND FIRST FACTORS: <-------- Output from subroutine GETDATA.
? 45,3     )
? 265,12   |
? 1899,9   |
? 34,2     |
? 273,3    } <------------------------------------ Input requested by subroutine GETDATA.
? 147,21   |
? 54,3     |
? 244,4    |
? 787,93   )
?
 *B #1, AT L.12                ------------------- FORTRAN CID PRINT command.  Undefined
? print *,list <--------------/                    values are printed as asterisks.
  45 3 ****************** 265 12 ****************** 1899 9
  ****************** 34 2 ****************** 273 3 ****************** 147
  21 ****************** 54 3 ****************** 244 4 ******************
  787 93 ****************** ****************** ******************
  ******************
? set,breakpoint,p.sortlst_l.1
 *WARN - LINE 1 NOT EXECUTABLE - LINE 6 WILL BE USED
 OK ? ok <---------------------------------------- Positive keyword is entered in
? go                                               response to warning message.
 *B #2, AT P.SORTLST_L.1
? set,trap,store,ntemp(1) [ <--------------------- A trap body command sequence is
 INTERPRET MODE TURNED ON <---------               entered.
 IN COLLECT MODE                    \
? print *,ntemp(1),' placed before ',list(1,j) --- STORE trap turns interpret mode
? ]                                                back on.
 END COLLECT
? set,breakpoint,l.10 [ <------------------------- A breakpoint body command sequence is
 IN COLLECT MODE                                   entered.
? print *,list(1,i),' placed at end'
? ]
 END COLLECT
? go
 *T #2, RJ IN L.8 <------------------------------- RJ trap is effective, because
? clear,trap,#2                                    interpret mode is on.
```

Figure 8-5. Debug Session B (Sheet 1 of 2)

```
? go
  265  PLACED AT END
  1899  PLACED AT END
  34  PLACED BEFORE 45
  273  PLACED BEFORE 1899                        Command Sequence Output.
  147  PLACED BEFORE 265
  54  PLACED BEFORE 147
  244  PLACED BEFORE 265
  787  PLACED BEFORE 1899
 *T #1, LINE AT P.FACTORS_L.10
? go
 *T #18, ABORT CPU ERROR EXIT 04 IN L.10 <------ Default ABORT trap occurs.
? display,#cpuerr
 #CPUERR = 4 <---------------------------------- ABORT trap occurred because of
? list,values,p.factors                          undefined value.
 P.FACTORS
 LIST(1,1) = 34,   LIST(2,1) = 2,   LIST(3,1) = 17,   LIST(1,2) = 45
 LIST(2,2) = 3,   LIST(3,2) = 15,   LIST(1,3) = 54,   LIST(2,3) = 3
 LIST(3,3) = 18,   LIST(1,4) = 147,   LIST(2,4) = 21,   LIST(3,4) = 7
 LIST(1,5) = 244,   LIST(2,5) = 4,   LIST(3,5) = 61,   LIST(1,6) = 265
 LIST(2,6) = 12,   LIST(3,6) = 0,   LIST(1,7) = 273,   LIST(2,7) = 3
 LIST(3,7) = 91,   LIST(1,8) = 787,   LIST(2,8) = 93,   LIST(3,8) = 0
 LIST(1,9) = 1899,   LIST(2,9) = 9,   LIST(3,9) = 211
 LIST(1,10) = -2882303760773920 94,   LIST(2,10) = -288230376077392093
 LIST(3,10) = -288230376077392092,   N = -288230376077392091 <--- N is undefined.
? n=9 <------------------------------------------ FORTRAN CID assignment command.
? clear,trap,* <--------------------------------- Clearing STORE trap turns off
 INTERPRET MODE TURNED OFF                        interpret mode.
? go,l.10
          NUMBER   FIRST    SECOND
                   FACTOR   FACTOR

            34        2       17
            45        3       15
            54        3       18
           147       21        7                 Output from subroutine OUTLST.
           244        4       61
           265       12   NON-INTEGER
           273        3       91
           787       93   NON-INTEGER
          1899        9      211
 *T #17, END IN L.11
? quit
 DEBUG TERMINATED
```

Figure 8-5. Debug Session B (Sheet 2 of 2)

```
/debug,on ◄──────────────────────────────────── Debug mode turned on before program
$DEBUG,ON.                                       compilation.
/ftn5,i=sesnc,lo=s/-a
1 FTN 5.0+528          80/08/29. 15.08.56 PAGE      1
     PROGRAM PROGA    76/176  OPT=0

        1        OVERLAY (OVFILE,0,0)
        2        PROGRAM PROGA (INPUT, OUTPUT)
        3        PRINT 100
        4 100    FORMAT (' IN (0,0) OVERLAY ')
        5 1      J=5
        6        CALL OVERLAY (6HOVFILE,2,0,6HRECALL)
        7 2      PRINT 200
        8 200    FORMAT (' RETURNED TO (0.0) OVERLAY')
        9        STOP
       10        END
1 FTN 5.0+528          80/08/29. 15.08.56 PAGE      1
     PROGRAM PROGX    76/176  OPT=0

        1        OVERLAY (OVFILE,2,0)
        2        PROGRAM PROGX
        3        PRINT 100
        4 100    FORMAT (' IN (2,0) OVERLAY ')
        5 1      K=7
        6        CALL OVERLAY (6HOVFILE,2,5,6HRECALL)
        7        PRINT 200
        8 200    FORMAT (' RETURNED TO (2,0) OVERLAY ')
        9        END
1 FTN 5.0+528          80/08/29. 15.08.56 PAGE      1
     PROGRAM PROGZ    76/176  OPT=0

        1        OVERLAY (OVFILE,2,5)
        2        PROGRAM PROGZ
        3 1      M=9
        4        PRINT 200
        5 200    FORMAT (' IN OVERLAY (2,5)' )
        6        END
     0.025 CP SECONDS COMPILATION TIME.
```

Figure 8-6. Session C Source Listing

```
/lgo
 CYBER INTERACTIVE DEBUG
? set auxiliary auxfile ewidbrt◄──────────Auxiliary file AUXFILE is created using
? list map                                 E, W, I, D, B, R, and T options.
 (0,0) * ,   (2,0),   (2,5)
? lm (0,0) (2,0) (2,5)
 (0,0),   DBUG.,   PROGA,   SYSAID=,   Q5NTRY=,   /FCL.C./,   /STP.END/
 /Q5.IO./,   CHMOVE=,   COMIO=,   /FCL=ENT/,   FCL=FDL,   FECMSK=
 FLTOUT=,   FMTAP=,   /AP.IO./,   FORSYS=,   FORUTL=,   GETFIT=,   KODER=
 OUTC=,   OUTCOM=,   OVERLAY,   QXPMD=,   /XJP.C./,   CPU.CPM,   CPU.CIO
 CPU.MVE,   CPU.SYS,   CMF.ALF,   CMM.CIA,   CMF.CSF,   CMM.FFA
 CMF.FRF,   CMF.GSS,   CMF.LDV,   CMF.LOV,   CMM.MEM,   CMM.R,   CMF.SLF
 FDL.RES,   /FDL.COM/,   FDL.MMI,   FOL.RES,   UCLOAD,   CTL$RM,   CTL$WR
 ERR$RM,   LIST$RM,   RM$SYS=
 (2,0),   PROGX
 (2,5),   PROGZ
? set trap overlay *
? d #home
 #HOME = (0,0)P.PROGA◄──────────Overlay included in location report.
? go
 IN (0,0) OVERLAY
 *T #1, OVERLAY (2,0) IN (2,0)P.PROGX_L.0
? d #home
 #HOME = (2,0)P.PROGX
? go l.3
 IN (2,0) OVERLAY
 *T #1, OVERLAY (2,5) IN (2,5)P.PROGZ_L.0
? d #home
 #HOME = (2,5)P.PROGZ
? list values
 (0,0)P.PROGA
 J = 5
 (2,0)P.PROGX
 K = 7
 (2,5)P.PROGZ
 M = -288230376077387186
? go l.3
 IN OVERLAY (2,5)
 RETURNED TO (2,0) OVERLAY
 RETURNED TO (0.0) OVERLAY
 *T #17, END IN (0,0)P.PROGA_L.9
? list values
 (0,0)P.PROGA
 J = 5
 (2,0)P.PROGX
 K = 7
 (2,5)P.PROGZ
 M = 9
? d #home
 #HOME = (0,0)P.PROGA
? list map
 (0,0) * ,   (2,0) * ,   (2,5) *
? quit
 DEBUG TERMINATED
```

Figure 8-7. Debug Session C

```
1                                                 CYBER INTERACTIVE DEBUG
0
 LIST MAP
 (0,0) * ,    (2,0),    (2,5)
 LM (0,0) (2,0) (2,5)
 (0,0),    DBUG.,    PROGA,    SYSAID=,    Q5NTRY=,    /FCL.C./,    /STP.END/
 /Q5.IO./,    CHMOVE=,    COMIO=,    /FCL=ENT/,    FCL=FDL,    FECMSK=
 FLTOUT=,    FMTAP=,    /AP.IO./,    FORSYS=,    FORUTL=,    GETFIT=,    KODER=
 OUTC=,    OUTCOM=,    OVERLAY,    QXPMD=,    /XJP.C./,    CPU.CPM,    CPU.CIO
 CPU.MVE,    CPU.SYS,    CMF.ALF,    CMM.CIA,    CMF.CSF,    CMM.FFA
 CMF.FRF,    CMF.GSS,    CMF.LDV,    CMF.LOV,    CMM.MEM,    CMM.R,    CMF.SLF
 FDL.RES,    /FDL.COM/,    FDL.MMI,    FOL.RES,    UCLOAD,    CTL$RM,    CTL$WR
 ERR$RM,    LIST$RM,    RM$SYS=
 (2,0),    PROGX
 (2,5),    PROGZ
 SET TRAP OVERLAY *
 D #HOME
 #HOME = (0,0)P.PROGA
 GO
 *T #1, OVERLAY (2,0) IN (2,0)P.PROGX_L.0
 D #HOME
 #HOME = (2,0)P.PROGX
 GO L.3
 *T #1, OVERLAY (2,5) IN (2,5)P.PROGZ_L.0
 D #HOME
 #HOME = (2,5)P.PROGZ
 LIST VALUES
 (0,0)P.PROGA
 J = 5
 (2,0)P.PROGX
 K = 7
 (2,5)P.PROGZ
 M = -288230376077387186
 GO L.3
 *T #17, END IN (0,0)P.PROGA_L.9
 LIST VALUES
 (0,0)P.PROGA
 J = 5
 (2,0)P.PROGX
 K = 7
 (2,5)P.PROGZ
 M = 9
 D #HOME
 #HOME = (0,0)P.PROGA
 LIST MAP
 (0,0) * ,    (2,0) * ,    (2,5) *
 QUIT
```

Figure 8-8. Session C Auxiliary File Listing

```
1              IDENTIFICATION DIVISION.
2              PROGRAM-ID.  SALES-TEST.
3             *    THIS PROGRAM TAKES SALES DATA FROM TWO YEARS TO STATISTICALLY
4             *    DETERMINE IF A NEW ADVERTISING METHOD INCREASED SALES.
5             *
6             *    INPUT DATA IS A LIST OF SALES OFFICES AND THEIR FIRST- AND LAST-
7             *    YEAR SALES FIGURES.  AN INPUT LINE IS AS FOLLOWS:
8             *      SALES-OFFICE              PICTURE X(20).
9             *      FIRST-YEAR-SALES          PICTURE 9999V99.
10            *      SECOND-YEAR-SALES         PICTURE 9999V99.
11            *
12            *    A DIFFERENCE STATISTIC IS CALCULATED FROM THE SALES FIGURES OF
13            *    EACH OFFICE BY DIVIDING THE FIRST-YEAR-SALES INTO THE DIFFERENCE OF
14            *    THE TWO SALES FIGURES FOR THE OFFICE.
15            *
16            *    A TABLE OF DIFFERENCE STATISTICS IS GIVEN TO THE FORTRAN SUBROUTINE
17            *    NORMAL WHICH PERFORMS THE STATISTICAL TEST.  NORMAL RETURNS THE
18            *    TEST RESULT THROUGH THE DATA-ITEM DECISION, AND THIS RESULT IS TRANS-
19            *    LATED INTO ONE OF THE FOLLOWING:
20            *      ACCEPT THE HYPOTHESIS THAT BOTH METHODS ARE EQUAL.
21            *      ACCEPT THE HYPOTHESIS THAT YEAR 2 METHOD IS BETTER.
22            *
23             ENVIRONMENT DIVISION.
24             CONFIGURATION SECTION.
25             SOURCE-COMPUTER.  CYBER-170.
26             OBJECT-COMPUTER.  CYBER-170.
27             INPUT-OUTPUT SECTION.
28             FILE-CONTROL.
29                 SELECT IN-FILE   ASSIGN TO "SALES".
30                 SELECT OUT-FILE   ASSIGN TO "OUTPUT".
31                 SELECT SORT-FILE   ASSIGN TO SORTFIL.
32             DATA DIVISION.
33             FILE SECTION.
34             FD IN-FILE
35                  BLOCK CONTAINS 640 CHARACTERS
36                 LABEL RECORD IS OMITTED
37                 DATA RECORD IS LINE-IN.
38             01  LINE-IN.
39                 05  OFFICE                 PICTURE X(20).
40                 05  FIRST-YEAR-SALES       PICTURE 9999V99.
41                 05  SECOND-YEAR-SALES      PICTURE 9999V99.
42                 05  FILLER                 PICTURE X(8).
43
44             FD OUT-FILE
45                 LABEL RECORD IS OMITTED
46                 DATA RECORDS ARE LINE-OUT, FINAL-LINE.
47             01  LINE-OUT.
48                 05  FILLER                 PICTURE X(10).
49                 05  OFFICE                 PICTURE X(20).
50                 05  FIRST-YEAR-SALES       PICTURE BB$$$$9.99.
51                 05  SECOND-YEAR-SALES      PICTURE BB$$$$9.99.
52                 05  DIFF-STAT              PICTURE -(9)9.9999.
53                 05  FILLER                 PICTURE X(50).
54             01  FINAL-LINE.
55                 05  FILLER                 PICTURE X(3).
56                 05  MEAN                   PICTURE ZZZZ9.9999.
57                 05  VARIANCE               PICTURE ZZZ9.9999.
58                 05  NUMBER-OF-OFFICES      PICTURE Z(7)9V.
59                 05  P-VALUE                PICTURE B(7)ZZZ9.9999.
60
61             SD  SORT-FILE
62                 RECORD CONTAINS 40 CHARACTERS
63                 DATA RECORD IS SORT-SALES-LINE.
64             01  SORT-SALES-LINE.
65                 05  OFFICE                 PICTURE X(20).
66                 05  FIRST-YEAR-SALES       PICTURE 9999V99.
67                 05  SECOND-YEAR-SALES      PICTURE 9999V99.
68                 05  FILLER                 PICTURE X(8).
69
```

Figure 8-9. Session D; COBOL Main Program (Sheet 1 of 3)

```
70            WORKING-STORAGE SECTION.
71            01  SPACE-LINE                  PICTURE X(100) VALUE SPACES.
72            01  HDG-1-LINE-1.
73                05  FILLER                  PICTURE X(12) VALUE SPACES.
74                05  FILLER                  PICTURE X(21)
75                                            VALUE "SALES OFFICE".
76                05  FILLER                  PICTURE X(40)
77                    VALUE "YEAR 1    YEAR 2          DIFFERENCE".
78            01  HDG-1-LINE-2.
79                05  FILLER                  PICTURE X(33) VALUE SPACES.
80                05  FILLER                  PICTURE X(40)
81                    VALUE "SALES     SALES           STATISTIC".
82            01  HDG-2-LINE-1.
83                05  FILLER                  PICTURE X(8) VALUE SPACES.
84                05  FILLER                  PICTURE X(7) VALUE "MEAN".
85                05  FILLER                  PICTURE X(10) VALUE "VARIANCE".
86                05  FILLER                  PICTURE X(15) VALUE "NUMBER OF".
87                05  FILLER                  PICTURE X(10) VALUE "P-VALUE".
88            01  HDG-2-LINE-2.
89                05  FILLER                  PICTURE X(26) VALUE SPACES.
90                05  FILLER                  PICTURE X(10) VALUE "OFFICES".
91            01  COMP-ITEMS.
92                05  DIFFERENCE-STAT-TABLE.
93                   10  DIFFERENCE-STAT  OCCURS 10 TIMES
94                                         INDEXED BY DIFF-INDEX.
95                      15  DIFFERENCE-STATISTIC  USAGE COMP-2.
96                05  NUMBER-OF-OFFICES       PICTURE 999V USAGE COMP-1.
97                05  ALPHA                   USAGE COMP-2.
98                05  P-VALUE                 USAGE COMP-2.
99                05  MEAN                    USAGE COMP-2.
100               05  VARIANCE                USAGE COMP-2.
101               05  DECISION                PICTURE 9V.
102
103           PROCEDURE DIVISION.
104           OPEN-AND-INITIALIZE SECTION.
105           OPEN-FILES.
106               OPEN INPUT IN-FILE
107                    OUTPUT OUT-FILE.
108           INITIALIZE-VALUES.
109               SET DIFF-INDEX TO 1.
110               MOVE ZERO TO NUMBER-OF-OFFICES OF COMP-ITEMS.
111
112           READ-DATA-AND-PERFORM-TEST SECTION.
113           SORT-THE-INPUT-RECORDS.
114               SORT SORT-FILE ON ASCENDING KEY OFFICE OF SORT-SALES-LINE
115                 INPUT PROCEDURE IS GET-INPUT-LINE
116                 OUTPUT PROCEDURE IS ARRANGE-AND-WRITE-SALES-DATA.
117
118           TEST-FOR-INCREASE-IN-SALES.
119
120               MOVE 0.05 TO ALPHA OF COMP-ITEMS.
121               ENTER FTN5 NORMAL
122                 USING NUMBER-OF-OFFICES OF COMP-ITEMS
123                       DIFFERENCE-STAT-TABLE
124                       ALPHA OF COMP-ITEMS
125                       MEAN OF COMP-ITEMS
126                       VARIANCE OF COMP-ITEMS
127                       P-VALUE OF COMP-ITEMS
128                       DECISION.
```

Figure 8-9. Session D; COBOL Main Program (Sheet 2 of 3)

```
129            WRITE LINE-OUT FROM SPACE-LINE.
130            WRITE LINE-OUT FROM HDG-2-LINE-1.
131            WRITE LINE-OUT FROM HDG-2-LINE-2.
132            WRITE LINE-OUT FROM SPACE-LINE.
133            MOVE CORRESPONDING COMP-ITEMS TO FINAL-LINE.
134            WRITE FINAL-LINE.
135            WRITE LINE-OUT FROM SPACE-LINE.
136            IF DECISION = 1 MOVE
137                "ACCEPT HYPOTHESIS THAT BOTH METHODS ARE EQUAL."
138                     TO LINE-OUT
139            ELSE MOVE
140                "ACCEPT HYPOTHESIS THAT YEAR 2 METHOD IS BETTER."
141                     TO LINE-OUT.
142            WRITE LINE-OUT.
143            WRITE LINE-OUT FROM SPACE-LINE.
144
145        END-PROCESSING-SECTION.
146        CLOSE-FILES.
147            CLOSE IN-FILE, OUT-FILE.
148        STOP-EXECUTION.
149            STOP RUN.
150
151        GET-INPUT-LINE SECTION.
152        READ-RECORD.
153            READ IN-FILE
154              AT END GO TO END-OF-DATA.
155            ADD 1 TO NUMBER-OF-OFFICES OF COMP-ITEMS.
156            RELEASE SORT-SALES-LINE FROM LINE-IN.
157            GO TO READ-RECORD.
158        END-OF-DATA.
159
160        ARRANGE-AND-WRITE-SALES-DATA SECTION.
161
162        WRITE-FIRST-HEADINGS.
163            WRITE LINE-OUT FROM SPACE-LINE.
164            WRITE LINE-OUT FROM HDG-1-LINE-1.
165            WRITE LINE-OUT FROM HDG-1-LINE-2.
166            WRITE LINE-OUT FROM SPACE-LINE.
167
168        GET-NEXT-SORT-FILE-RECORD.
169            RETURN SORT-FILE RECORD
170              AT END GO TO END-OF-SECTION.
171
172        CALCULATE-DIFFERENCE-STATISTIC.
173            COMPUTE  DIFFERENCE-STATISTIC (DIFF-INDEX)
174                     = (SECOND-YEAR-SALES OF SORT-SALES-LINE
175                     -  FIRST-YEAR-SALES OF SORT-SALES-LINE) /
176                        FIRST-YEAR-SALES OF SORT-SALES-LINE.
177
178        ARRANGE-OUTPUT-LINE.
179            MOVE CORRESPONDING SORT-SALES-LINE TO LINE-OUT.
180            MOVE DIFFERENCE-STATISTIC (DIFF-INDEX) TO DIFF-STAT.
181            SET DIFF-INDEX UP BY 1.
182            WRITE LINE-OUT.
183            GO GET-NEXT-SORT-FILE-RECORD.
184        END-OF-SECTION.
```

Figure 8-9. Session D; COBOL Main Program (Sheet 3 of 3)

```
1 FTN 5.1+538        81/07/15. 14.40.02 PAGE     1
  SUBROUTINE NORMAL   74/74   OPT=0

          1        SUBROUTINE NORMAL (N,X,ALPHA,AVG,VAR,PVALUE,DECISN)
          2 C     THIS SUBROUTINE USES THE NORMAL DISTRIBUTION TO TEST
          3 C     HYPOTHESIS HO:  POPULATION MEAN IS EQUAL TO ZERO
          4 C         VERSUS HA:  POPULATION MEAN IS GREATER THAN ZERO.
          5 C
          6 C     VALUES PASSED TO THIS SUBROUTINE ARE:
          7 C      N        SAMPLE POPULATION SIZE
          8 C      X        SAMPLE POPULATION ARRAY
          9 C      ALPHA    SIGNIFICANCE LEVEL OF THE TEST
         10 C
         11 C     VALUES RETURNED BY THIS SUBROUTINE ARE:
         12 C      AVG      SAMPLE MEAN (AVERAGE)
         13 C      VAR      VARIANCE
         14 C      PVALUE   P-VALUE -- APPROXIMATED THROUGH FUNCTION P(Y)
         15 C      DECISN   TEST RESULT -- 1=ACCEPT HO; 2=ACCEPT HA
         16 C
         17        DIMENSION X(10)
         18        INTEGER DECISN
         19        P(Y) = .5 * (1 + .196854*Y + .115194*Y**2
         20       A                 + .000344*Y**3 + .019527*Y**4)
         21       B                 **(-4)
         22 C     CALCULATE MEAN AND VARIANCE
         23        AVG=0
         24        VAR=0
         25        DO 100 I=1,N
         26        AVG=AVG+X(I)
         27        VAR=VAR+X(I)**2
         28 100    CONTINUE
         29        AVG=AVG/N
         30        VAR=VAR/N-AVG**2
         31 C     GET PVALUE
         32        Z=(AVG*SQRT(REAL(N)))/SQRT(VAR)
         33        IF (Z .GE. 0) THEN
         34             PVALUE = P(Z)
         35          ELSE
         36             PVALUE = 1 - P(-Z)
         37        END IF
         38 C     ACCEPT HO OR HA
         39        IF (PVALUE .LT. ALPHA) THEN
         40             DECISN = 2
         41            ELSE
         42             DECISN = 1
         43        END IF
         44        RETURN
         45        END
```

Figure 8-10. Session D; FORTRAN Subroutine

```
BOSTON          347988405772
SEATTLE         472359417762
DENVER          246836156618
LOS ANGELES     520805352443
DALLAS          662118477667
MIAMI           322441322001
CHICAGO         900732898254
SALT LAKE       227400225367
```

Figure 8-11. Session D; Input Data on File SALES

```
/lgo

            SALES OFFICE          YEAR 1     YEAR 2        DIFFERENCE
                                  SALES      SALES         STATISTIC

          BOSTON                $3479.88   $4057.72        0.1660
          CHICAGO               $9007.32   $8982.54       -0.0027
          DALLAS                $6621.18   $4776.67       -0.2785
          DENVER                $2468.36   $1566.18       -0.3654
          LOS ANGELES           $5208.05   $3524.43       -0.3232
          MIAMI                 $3224.41   $3220.01       -0.0013
          SALT LAKE             $2274.00   $2253.67       -0.0089
          SEATTLE               $4723.59   $4177.62       -0.1155

        MEAN   VARIANCE  NUMBER OF      P-VALUE
                          OFFICES

       0.1162   0.0310      8           0.9691

ACCEPT HYPOTHESIS THAT YEAR 2 METHOD IS BETTER.

LGO.
/debug,on
DEBUG,ON.
/lgo
 CYBER INTERACTIVE DEBUG
? set,trap,procedure,* [
 IN COLLECT MODE
? display,#proc
? ]
 END COLLECT
? set,breakpoint,pr.get-input-line
? go
 #PROC = P.SALES-T_PR.OPEN-AND-INITIALIZE
 #PROC = P.SALES-T_PR.OPEN-FILES
 #PROC = P.SALES-T_PR.INITIALIZE-VALUES
 #PROC = P.SALES-T_PR.READ-DATA-AND-PERFORM-TEST
 #PROC = P.SALES-T_PR.SORT-THE-INPUT-RECORDS
 *B #1, AT PR.GET-INPUT-LINE
? clear,breakpoint,*
? step,1,procedure
 #PROC = P.SALES-T_PR.GET-INPUT-LINE
 *S PROCEDURE AT PR.READ-RECORD
? step
 *S PROCEDURE AT PR.READ-RECORD
? display line-in
 BOSTON              347988405772

? step
 *S PROCEDURE AT PR.READ-RECORD
? display line-in
 SEATTLE             472359417762
```

Annotations:

- (table output) Programs compiled for use with CID are run with debug mode turned off. An error exists: The test concludes that the year 2 method is better even though all but one office lost money.
- /debug,on — Debug mode is turned on.
- set,trap,procedure,* [— PROCEDURE trap with a trap body is set.
- set,breakpoint,pr.get-input-line — Breakpoint set at beginning of GET-INPUT-LINE section.
- #PROC lines — As a result of the PROCEDURE trap, debug variable #PROC is automatically displayed at the beginning of each paragraph or section in the Procedure Division.
- *B #1, AT PR.GET-INPUT-LINE — Breakpoint suspends execution.
- step,1,procedure — STEP command is entered.
- step — STEP command with no parameters repeats previous STEP.
- display line-in — COBOL CID DISPLAY command is entered.

Figure 8-12. Debug Session D (Sheet 1 of 3)

```
? set,breakpoint,pr.end-of-data
? go
 #PROC = P.SALES-T_PR.READ-RECORD
 #PROC = P.SALES-T_PR.READ-RECORD
 #PROC = P.SALES-T_PR.READ-RECORD
 #PROC = P.SALES-T_PR.READ-RECORD
 #PROC = P.SALES-T_PR.READ-RECORD
 #PROC = P.SALES-T_PR.READ-RECORD
 *B #1, AT PR.END-OF-DATA
?

? clear,breakpoint,*
? clear,trap,*
? set,breakpoint,p.normal_l.1
 *WARN - LINE 1 NOT EXECUTABLE - LINE 23 WILL BE USED
 OK ? ok   <-- Positive keyword is entered in response
               to warning message.
? go

            SALES OFFICE          YEAR 1    YEAR 2       DIFFERENCE
                                  SALES     SALES        STATISTIC

           BOSTON                $3479.88  $4057.72        0.1660
           CHICAGO               $9007.32  $8982.54       -0.0027
           DALLAS                $6621.18  $4776.67       -0.2785
           DENVER                $2468.36  $1566.18       -0.3654
           LOS ANGELES           $5208.05  $3524.43       -0.3232
           MIAMI                 $3224.41  $3220.01       -0.0013
           SALT LAKE             $2274.00  $2253.67       -0.0089
           SEATTLE               $4723.59  $4177.62       -0.1155
 *B #1, AT P.NORMAL_L.1   <-- Breakpoint suspends execution in
                              FORTRAN subroutine.
? print *,n,alpha   <-- FORTRAN CID PRINT command is entered.
  8 5.E-02
? print *,x
   .1660517029323 -2.751095775436E-03 -.2785772324571 -.3654977393897
  -.3232726260309 -1.3645907313276E-03 -8.9401934916447E-03
  -.1155836979924 -7.1738305547276E+57 -7.1738305547276E+57
? set,breakpoint,p.normal_l.44
? go
 *B #2, AT P.NORMAL_L.44
? print *,pvalue,decisn
   .9691398559997 1   <-- DECISN=1.  Hypothesis that both methods
                          are equal should be accepted.
? set,trap,line,*
? go
 *T #1, LINE AT L.44
? go
 *T #1, LINE AT P.SALES-T_L.129   <-- Return to COBOL main program.
? display decision of comp-items   <-- DECISION is blank instead of 1.  A look
                                       at the source listings shows that
                                       DECISION is not a COMP-2 item, even
                                       though DECISN, in the FORTRAN subroutine,
                                       is floating-point.  DECISION should be
                                       made COMP-2.
```

Figure 8-12. Debug Session D (Sheet 2 of 3)

```
? list,values
 P.SALES-T
 <01>LINE-IN:<05>OFFICE=SALT LAKE           <05>FIRST-YEAR-SALES= 2274.00
 <05>SECOND-YEAR-SALES= 2253.67<01>LINE-OUT
 <05>OFFICE=SEATTLE                <05>FIRST-YEAR-SALES=  $4723.59
 <05>SECOND-YEAR-SALES=  $4177.62<05>DIFF-STAT=          -0.1155
 <01>FINAL-LINE:<05>MEAN=          SEA<05>VARIANCE=TTLE
 <05>NUMBER-OF-OFFICES=          <05>P-VALUE=  $4723.59  $417
 <01>SORT-SALES-LINE:<05>OFFICE=SEATTLE
 <05>FIRST-YEAR-SALES= 4723.59<05>SECOND-YEAR-SALES= 4177.62
 <01>SPACE-LINE=

                              <01>HDG-1-LINE-1:<01>HDG-1-LINE-2
 <01>HDG-2-LINE-1:<01>HDG-2-LINE-2:<01>COMP-ITEMS
 <05>DIFFERENCE-STAT-TABLE:<10>DIFFERENCE-STAT:<IX>DIFF-INDEX= 9
 <15>DIFFERENCE-STATISTIC[1]=+.016605170293228E+0001[2]=
 -.027510957754359E-0001[3]=-.027857723245705E+0001[4]=
 -.036549773938971E+0001[5]=-.032327262603085E+0001[6]=
 -.013645907313275E-0001[7]=-.089401934916446E-0001[8]=
 -.011558369799241E+0001[9]=-.717383055472760E+0058[10]=
 -.717383055472760E+0058<05>NUMBER-OF-OFFICES= 8<05>ALPHA=
 +.049999999999999E+0000<05>P-VALUE=+.096913985599972E+0001
 <05>MEAN=-.011624193411702E+0001<05>VARIANCE=+.031077960211124E+0000
 <05>DECISION=
 P.NORMAL
 ALPHA = .50000000000000E-01,    AVG = -.11624193411702,   DECISN = 1
 I = 9,   N = 8,   PVALUE = .96913985599973,   VAR = .31077960211124E-01
 X(1) = .16605170293228,   X(2) = -.27510957754360E-02
 X(3) = -.27857723245705,   X(4) = -.36549773938972
 X(5) = -.32327262603085,   X(6) = -.13645907313276E-02
 X(7) = -.89401934916447E-02,    X(8) = -.11558369799242
 X(9) = -.71738305547276E+58,    X(10) = -.71738305547276E+58,   Y = -I
 Z = -1.8650117305834
? clear,trap,*
? go

        MEAN   VARIANCE  NUMBER OF       P-VALUE
                          OFFICES

       0.1162   0.0310       8           0.9691

ACCEPT HYPOTHESIS THAT YEAR 2 METHOD IS BETTER.

 *T #17, END IN L.149
? quit
 DEBUG TERMINATED
```

LIST,VALUES output for COBOL main program.

LIST, VALUES output for FORTRAN subroutine.

Default END trap.

Figure 8-12. Debug Session D (Sheet 3 of 3)

STANDARD CHARACTER SETS A

Control Data operating systems offer the following variations of a basic character set:

- CDC 64-character set
- CDC 63-character set
- ASCII 64-character set
- ASCII 63-character set

The set in use at a particular installation is specified when the operating system is installed.

Depending on another installation option, the system assumes an input deck has been punched either in 026 or in 029 mode (regardless of the character set in use). Under NOS/BE, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of the job statement or any 7/8/9 card. The specified mode remains in effect throughout the job unless it is reset by specification of the alternate mode on a subsequent 7/8/9 card.

Under NOS, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of any 6/7/9 card, as described above for a 7/8/9 card. In addition, 026 mode can be specified by a card with 5/7/9 multipunched in column 1; 029 mode can be specified by a card with 5/7/9 multipunched in column 1 and a 9 punched in column 2.

Graphic character representation appearing at a terminal or printer depends on the installation character set and the terminal type. Characters shown in the CDC Graphic column of the standard character set table (table A-1) are applicable to BCD terminals; ASCII graphic characters are applicable to ASCII-CRT and ASCII-TTY terminals.

TABLE A-1. STANDARD CHARACTER SETS

Display Code (octal)	CDC			ASCII		
	Graphic	Hollerith Punch (026)	External BCD Code	Graphic Subset	Punch (029)	Code (octal)
00†	: (colon)††	8-2	00	: (colon)††	8-2	072
01	A	12-1	61	A	12-1	101
02	B	12-2	62	B	12-2	102
03	C	12-3	63	C	12-3	103
04	D	12-4	64	D	12-4	104
05	E	12-5	65	E	12-5	105
06	F	12-6	66	F	12-6	106
07	G	12-7	67	G	12-7	107
10	H	12-8	70	H	12-8	110
11	I	12-9	71	I	12-9	111
12	J	11-1	41	J	11-1	112
13	K	11-2	42	K	11-2	113
14	L	11-3	43	L	11-3	114
15	M	11-4	44	M	11-4	115
16	N	11-5	45	N	11-5	116
17	O	11-6	46	O	11-6	117
20	P	11-7	47	P	11-7	120
21	Q	11-8	50	Q	11-8	121
22	R	11-9	51	R	11-9	122
23	S	0-2	22	S	0-2	123
24	T	0-3	23	T	0-3	124
25	U	0-4	24	U	0-4	125
26	V	0-5	25	V	0-5	126
27	W	0-6	26	W	0-6	127
30	X	0-7	27	X	0-7	130
31	Y	0-8	30	Y	0-8	131
32	Z	0-9	31	Z	0-9	132
33	0	0	12	0	0	060
34	1	1	01	1	1	061
35	2	2	02	2	2	062
36	3	3	03	3	3	063
37	4	4	04	4	4	064
40	5	5	05	5	5	065
41	6	6	06	6	6	066
42	7	7	07	7	7	067
43	8	8	10	8	8	070
44	9	9	11	9	9	071
45	+	12	60	+	12-8-6	053
46	-	11	40	-	11	055
47	*	11-8-4	54	*	11-8-4	052
50	/	0-1	21	/	0-1	057
51	(	0-8-4	34	(	12-8-5	050
52	)	12-8-4	74	)	11-8-5	051
53	$	11-8-3	53	$	11-8-3	044
54	=	8-3	13	=	8-6	075
55	blank	no punch	20	blank	no punch	040
56	, (comma)	0-8-3	33	, (comma)	0-8-3	054
57	. (period)	12-8-3	73	. (period)	12-8-3	056
60	≡	0-8-6	36	#	8-3	043
61	[	8-7	17	[	12-8-2	133
62	]	0-8-2	32	]	11-8-2	135
63	%††	8-6	16	%††	0-8-4	045
64	≠	8-4	14	" (quote)	8-7	042
65	↱	0-8-5	35	_ (underline)	0-8-5	137
66	∨	11-0	52	!	12-8-7	041
67	∧	0-8-7	37	&	12	046
70	↑	11-8-5	55	' (apostrophe)	8-5	047
71	↓	11-8-6	56	?	0-8-7	077
72	<	12-0	72	<	12-8-4	074
73	>	11-8-7	57	>	0-8-6	076
74	≤	8-5	15	@	8-4	100
75	≥	12-8-5	75	\	0-8-2	134
76	¬	12-8-6	76	¬ (circumflex)	11-8-7	136
77	; (semicolon)	12-8-7	77	; (semicolon)	11-8-6	073

†Twelve zero bits at the end of a 60-bit word in a zero byte record are an end-of-record mark rather than two colons.

††In installations using a 63-graphic set, display code 00 has no associated graphic or card code; display code 63 is the colon (8-2 punch). The % graphic and related card codes do not exist and translations yield a blank (55_8).

DIAGNOSTICS B

Diagnostic messages issued by CYBER Interactive Debug are divided into the following categories:

- Error messages
- Warning messages
- Informative messages

ERROR MESSAGES

Error messages issued by CYBER Interactive Debug are listed in table B-1. These messages are issued in one of the following forms:

```
    *ERROR - message text
    ?
or
    *CMD - (command text) *ERROR - message text
    ?
```

When in collect mode, errors, such as those involving invalid syntax, are detected and reported prior to being collected, thus allowing them to be corrected at that time. Other errors are not detected until execution of the command is attempted.

Refer to section 7 for a discussion of error processing, including descriptions of available user responses to error messages, and the resulting actions.

WARNING MESSAGES

Warning messages issued by CYBER Interactive Debug are listed in table B-2. These messages have one of the following forms:

```
    *WARN - message text
    OK?
or
    *CMD - (command text) *WARN - message text
    OK?
```

Most of the warning messages indicate by their wording the action that is taken if you respond with a positive acknowledgment (YES, ACCEPT, or OK, as described in section 7).

Warning messages can be suppressed by issuing a SET,OUTPUT command that does not include the W parameter in its option list. The action indicated in the message automatically occurs. (The SET,OUTPUT command is described in section 5.)

INFORMATIVE MESSAGES

Informative messages issued by CYBER Interactive Debug are listed in table B-3. These messages have the form:

```
    message text
```

Informative messages indicate the following: changes in the status of CID, changes in the status of commands that process a list, commands which confirm specific actions taken, conditions when a list element cannot be processed, and conditions when there is no action to be taken.

After the informative message is issued, CYBER Interactive Debug does not pause for a response, except when the message announces the start or resumption of a debug session. Any remaining elements in a list are processed after reporting a list element that cannot be processed.

TABLE B-1. ERROR MESSAGES

Message	Significance	Action
*ERROR - ADDRESS IN ECS/LCM	GO or EXECUTE has been supplied with a location parameter that is in extended memory. If supplied, the address must be one in central memory.	Correct and reenter.
*ERROR - ADDRESS IN UNLOADED OVERLAY	A specified symbolic address implies one contained in an overlay not currently loaded. LIST,MAP indicates which overlays are currently loaded by displaying an asterisk following the overlay designation.	Confine symbolic addresses to those in currently loaded overlays.
*ERROR - ADDRESS OUTSIDE USER AREA	An address reference is to a location in DBUG. beyond the first 100_8 (approximately) locations or beyond the field length. You cannot access these locations.	Reenter with an allowable address.

TABLE B-1. ERROR MESSAGES (Contd)

Message	Significance	Action
*ERROR - ARGUMENT LEVEL MISMATCH	Declared level of actual argument does not agree with level of FORTRAN 5 subroutine or function format argument.	Correct FORTRAN program.
*ERROR - COMMAND/EXPRESSION TOO COMPLEX	An expression is too complex in an IF, PRINT, or assignment statement; or a PRINT list is too complex.	Simplify and reenter.
*ERROR - CONTROL INDEX xxxx DOUBLY DEFINED, REASSIGN INDEX	The same control index has been used more than once in the nested implied DO loops of the PRINT statement.	Correct and reenter.
*ERROR - COUNT TOO LARGE	The count in a STEP command is greater than 536870911.	Reenter with a lower count.
*ERROR - DEBUG INTERNAL ERROR	An error in CID is preventing further processing. CID must be aborted. The program being debugged could have damaged a portion of DBUG.	Try a new debug session with all execution performed in interpret mode, which protects DBUG. code.
*ERROR - ENTRY POINT xxxx NOT CALLED	A TRACEBACK has been requested from an entry point to which no call has been made. LIST,MAP,P.progname lists all entry points in a given program module.	Try tracing from each named entry point.
*ERROR - ENTRY POINT xxxx NOT LOADED	TRACEBACK cannot proceed because the specified entry point is contained in an overlay not currently loaded.	Enter another command.
*ERROR - ERROR MESSAGES MAY NOT BE SUPPRESSED	Completing the current SET,OUTPUT, SET,AUXILIARY, CLEAR,OUTPUT, or CLEAR,AUXILIARY command results in no file being designated to receive error messages.	Assign file options consistent with this restriction.
*ERROR - FIRST ADDRESS OF RANGE GREATER THAN SECOND	An address range in the format of the ellipsis notation is invalid for the reason stated.	Correct and reenter.
*ERROR - FORMAT CODE MUST BE A, C, D, I, F, or O	DISPLAY code is incorrect.	Reenter with a valid code, or omit the DISPLAY code allowing the default value to be used.
*ERROR - ILLEGAL MASKING OR LOGICAL OPERAND	An expression in an IF, PRINT, or assignment statement contains an AND or OR operator with one type logical operand and one operand other than type logical.	Correct and reenter.
*ERROR - ILLEGAL TYPE	A COBOL data item is being used in a way that is wrong for its type.	Correct and reenter.
*ERROR - ILLEGAL TYPE FOR INDEX xxxx yyyy VALUE	An initial final or increment value (as specified by yyyy) of index xxxx in an implied DO of a PRINT statement has a data type other than integer, real, or double-precision.	Correct and reenter.
*ERROR - IMPROPERLY NEGATIVE	A COBOL literal or data item is negative and is being used where only positive numbers are allowed.	Correct and reenter.
*ERROR - INCOMPATIBLE OPERAND TYPES	Expression contains incompatible operands, usually a string and a nonstring.	Correct and reenter.
*ERROR - INVALID FIRST, LAST OR STEP VALUE	A breakpoint has been supplied with an invalid frequency parameter.	Check that all such parameters are positive. Check that LAST is not less than FIRST. Reenter with corrected values.

TABLE B-1. ERROR MESSAGES (Contd)

Message	Significance	Action
*ERROR - INVALID PARAMETER xxxx	The supplied HELP parameter is invalid.	Enter HELP * for a list of valid parameters. Reenter the HELP command with a valid parameter.
*ERROR - INVALID PARAMETER TYPE xxxx	The parameter supplied is a type not allowed for this command. If this error occurs in a command sequence body, it is detected at collect time.	Reenter with a correct parameter type.
*ERROR - INVALID QUALIFIER FOR OVERLAY TRAPS	An overlay trap qualifier other than an overlay designation for * was specified in a SET,TRAP, LIST,TRAP, CLEAR,TRAP, or SAVE,TRAP command.	Correct and reenter.
*ERROR - INVALID SYNTAX xxxx	A recognizable command contains syntactic elements of a form or in an order other than as defined for the command. If this error occurs in a command sequence body, it is detected at collect time.	Enter HELP followed by the command name to obtain the valid syntax for the command. Correct and reenter.
*ERROR - INVALID TRAP TYPE xxxx	A SET,TRAP command has an invalid trap type parameter value.	Reenter with a valid trap type.
*ERROR - INVALID TYPE FOR STEP - xxxx	The type parameter in a STEP command is invalid.	Reenter with a valid type.
*ERROR - LABEL LONGER THAN 7 CHARACTERS xxxx	A LABEL or JUMP command is supplied with a label parameter greater than seven characters.	Reenter with a valid label.
*ERROR - LINE NUMBERS NOT AVAILABLE	Referenced program is not a high level language program compiled for use with CID.	Check the home program.
*ERROR - MESSAGE LEVEL CODE MUST BE L, P, S, or PR	The message level code for a SET,TRAP command is other than the L, P, S, or PR parameter. If the parameter is omitted, the default value is used.	Reenter with a valid code for the parameter, or omit the parameter.
*ERROR - MOVE CANT DO THIS	A COBOL CID MOVE command has an illegal combination of types.	Correct and reenter.
*ERROR - NAME LONGER THAN 7 CHARACTERS xxxx	The supplied name of a program module, common block, entry point, group, or file exceeds seven characters in length. If this error occurs in a command sequence body, it is detected at collect time.	Correct and reenter.
*ERROR - NAME LONGER THAN 30 CHARACTERS xxxx	The name of a COBOL data item, supplied in a CID commmand, is longer than 30 characters.	Correct and reenter.
*ERROR - NO COMMON BLOCK xxxx	A reference has been made to a common block xxxx which does not exist; or if an overlay qualifier has been supplied, the common block is not in that overlay.	Check spelling; correct and reenter.
*ERROR - NO ENTRY POINT xxxx	A reference has been made to an entry point name xxxx which does not exist; or if an overlay qualifier has been supplied, the entry point is not in that overlay.	Check spelling or overlay qualifier; correct and reenter.
*ERROR - NO EXECUTABLE STATEMENT n	An attempt was made to reference a FORTRAN statement with label n. No such statement exists in the referenced program, or the statement is nonexecutable.	Check the program listing; correct and reenter.
*ERROR - NO FILE OR GROUP xxxx	The file or group named in a READ parameter does not exist.	Check spelling; check to see if the file is logically connected to the job.

TABLE B-1. ERROR MESSAGES (Contd)

Message	Significance	Action
*ERROR - NO LABEL xxxx	A JUMP command has referenced a label which does not exist in the current command sequence	Correct the command sequence accordingly.
*ERROR - NO LOOKUP IMPLEMENTED	Language not supported by CID; therefore, symbol table lookup is not implemented.	Check source language of the home program.
*ERROR - NO OVERLAYS	An overlay reference has been made in a non-overlay environment. This error is detected at collect time if it occurs in a command sequence body or if a specific overlay is referenced.	Confine CID commands and address qualifiers to those acceptable in a nonoverlay environment.
*ERROR - NO OVERLAY (xxxx)	The specified overlay does not exist. LIST,MAP indicates all existing overlays.	Reenter with the corrected overlay designation.
*ERROR - NO PROCEDURE NAME xxxx	An attempt was made to reference procedure name xxxx, but no such procedure name exists in the procedure division of the home program.	Check spelling of procedure name and home program designation.
*ERROR - NO PROGRAM xxxx	A reference has been made to a program module xxxx which does not exist; or if an overlay qualifier is supplied, the program module is not in that overlay.	Correct and reenter.
*ERROR - NO PROGRAM VARIABLE xxxx	An attempt was made to reference variable xxxx but no such variable exists in the referenced or home program. LIST,VALUES lists the names and values of all loaded variables.	Check spelling and home program.
*ERROR - xxxx NOT INTEGER	A COBOL CID command contains a subscript or reference modification that has digits to the right of the decimal point.	Reenter with integer subscript or reference modification.
*ERROR - OPTION CODE MUST BE B, D, E, I, R, T, OR W	An invalid option code was specified in the option list of a SET,OUTPUT or SET,AUXILIARY command.	Reenter with all valid option codes.
*ERROR - xxxx OUTSIDE -131071 TO 131071	A subscript or reference modification in a COBOL CID command is less than -131071 or greater than 131071.	Reenter with a valid subscript or reference modification.
*ERROR - PARAMETER MUST BE NORMAL OR ABNORMAL	An invalid parameter was supplied with QUIT.	Reenter with NORMAL or ABNORMAL.
*ERROR - PARAMETER MUST BE ON OR OFF	Valid parameter values for SET,INTERPRET are ON and OFF.	Reenter with a valid parameter value.
*ERROR - PARAMETER REFERENCED BEFORE FIRST SUBROUTINE CALL	A reference has been made to a formal argument of a FORTRAN subroutine or function prior to ever reaching its first executable statement. In this circumstance, the address of any formal parameter is unknown, and its value is undefined (as indicated in LIST,VALUES).	Allow execution to proceed until entry into the routine has been reached before referencing formal parameters.
*ERROR - PROGRAM xxxx NOT CALLED	TRACEBACK cannot proceed because an entry point in the specified program has never been called.	Enter another command.
*ERROR - PROGRAM xxxx NOT LOADED	TRACEBACK cannot proceed because the program being referenced is in an overlay which is not currently loaded.	Enter another command.
*ERROR - PROGRAM HAS COMPLETED	An attempt has been made with either GO or EXECUTE to continue program execution from the point where program termination has been reached.	Reenter specifying some other execution address, or issue QUIT.

TABLE B-1. ERROR MESSAGES (Contd)

Message	Significance	Action
*ERROR - RANGE ADDRESSES IN DIFFERENT MEMORIES	An address range in the form address 1...address 2 has a CM address and an extended memory address. Both addresses must be for the same memory. An indirect reference is assumed to be a CM reference.	Correct and reenter.
*ERROR - RANGE OF DEBUG VARIABLES NOT ALLOWED	An address range in the form #xx...#yy has been encountered. This form is not valid, since #xx and #yy alone represent variables rather than addresses.	Replace with #xx+0...#yy+0.
*ERROR - REBUILD FILE ZZZZZDT	CID is unable to use debug tables from old ZZZZZDT file.	Make sure that the release level of LOADER is the same as CID. Reload program to create new ZZZZZDT file.
*ERROR - RECURSIVE CALL TO ENTRY POINT xxxx	TRACEBACK has encountered the same entry point for a second time. Program logic flow is in error.	Direct debugging efforts towards locating the logic flow error.
*ERROR - RECURSIVE READ OF xxxx	The group or file named in the current READ parameter is a nested command sequence or the current sequence.	Redesign sequence logic to avoid this situation.
*ERROR - REFERENCE MOD OUT OF RANGE	A reference modification in a COBOL CID command is zero or greater than the size of the data item.	Reenter with a valid reference modification.
*ERROR - RELATION CODE MUST BE EQ, NE, GT, GE, LT, OR LE	The SKIPIF relational operator must be EQ, NE, GT, GE, LT, or LE.	Reenter with a valid code.
*ERROR - RELATIVE ADDRESS OUTSIDE BLOCK	A module offset is equal to or greater than its length. LIST,MAP gives the program length.	Check for the missing octal suffix B on the offset value if octal was intended. Correct and reenter.
*ERROR - RESPONSE QUALIFIER MUST BE LINE OR SEQ	In response to an error, warning, veto, or interrupt of a command sequence, a response keyword has been followed by text beginning with other than LINE, SEQ, or : (colon).	Enter any desired valid response.
*ERROR - SET CANT DO THIS	A COBOL CID SET command has an illegal combination of types.	Correct and reenter.
*ERROR - STATEMENT LABELS NOT AVAILABLE	Referenced program is not a high level program compiled for use with CID.	Check the home program.
*ERROR - STRING/CONSTANT TOO LONG	A character string in a FORTRAN arithmetic expression exceeds 41 characters.	Correct and reenter.
*ERROR - SUBSCRIPT OUT OF RANGE	Subscript for BASIC variable or COBOL data item is out of range.	Correct and reenter.
*ERROR - TOO FEW PARAMETERS	At least one additional parameter is required. If this error occurs in a command sequence body, it is detected at collect time. Some commands do not default the last parameter, but require an asterisk.	Where other than a last parameter is being defaulted, indicate by using two consecutive commas. Check the syntax of the command, and reenter.
*ERROR - TOO MANY BREAKPOINTS	The number of breakpoints has reached the maximum allowed.	One or more existing breakpoints must be cleared before any new ones can be set.
*ERROR - TOO MANY GROUPS	The number of groups has reached the maximum allowed.	One or more existing groups must be cleared before any new ones can be set.

TABLE B-1. ERROR MESSAGES (Contd)

Message	Significance	Action
*ERROR - TOO MANY NESTED COMMAND SEQUENCES	The number of nested command sequences has reached the maximum allowed. A READ or PAUSE command is not allowed until the current sequence is terminated and the previous sequence is resumed.	Enter GO to resume the previous sequence immediately.
*ERROR - TOO MANY PARAMETERS	Too many parameters have been supplied. If this error occurs in a command sequence body, it is detected at collect time.	Correct and reenter.
*ERROR - TOO MANY TRACE LEVELS	The TRACEBACK output has reached its built-in feasibility limit. Program logic flow could have errors.	Correct and reenter.
*ERROR - TOO MANY TRAPS	The number of traps has reached the maximum allowed.	Clear one or more existing traps before setting new ones.
*ERROR - TOO MANY/FEW SUBSCRIPTS	Incorrect number of subscripts used in an array element reference.	Correct and reenter.
*ERROR - UNKNOWN COMMAND	The command text does not contain a syntactically recognizable command name. If this error occurs in a command sequence body, it is detected at collect time. HELP,CMDS list all valid command names.	Check spelling. Reenter a valid command.
*ERROR - VALUE OUT OF RANGE OR INDEFINITE	A FORTRAN expression in an IF, PRINT, or assignment statement has evaluated to an out of range or indefinite result. That is, overflow has occurred.	Correct and reenter.
*ERROR - VARIABLE NAMES NOT AVAILABLE	Referenced program is not a high level program compiled for use with CID.	Check the home program.
*ERROR - VERY LONG CONSTANT	Constant in MOVE, ENTER, or DISPLAY command is too long.	Correct and reenter.
*ERROR - WRITING INTO RA+1	Attempt was made to set breakpoint at RA+1.	Correct and reenter.
*ERROR - ZERO INCREMENT FOR INDEX xxxx IN DO LOOP	The increment for the indicated index of an implied DO loop in a PRINT list is zero. Both positive and negative values are valid, allowing subscripted variable elements to be printed in ascending or descending subscript order.	Correct and reenter.
*ERROR - ZERO OR NEGATIVE COUNT	The count parameter of a DISPLAY, ENTER, MOVE, or STEP command is zero or negative.	Reenter with a positive value.

TABLE B-2. WARNING MESSAGES

Message	Significance
*WARN - ADDRESS RANGE WILL BE TRUNCATED	An address range for ENTER, DISPLAY, or MOVE extends beyond the user field length or into DBUG. beyond the first 100 (approximately) locations.
*WARN - ALL WILL BE CLEARED	A CLEAR,TRAP, CLEAR,BREAKPOINT, or CLEAR,GROUP command has been issued with no parameters.

TABLE B-2. WARNING MESSAGES (Contd)

Message	Significance
*WARN - ARGUMENT LENGTHS DISAGREE - SUBROUTINES WILL BE USED	Length of FORTRAN 5 type character formal argument does not agree with length of actual character string passed. Problem can be a program error.
*WARN - BREAKPOINT WILL BE SET AT ENTRY POINT	This warning is issued only if the specified address is an entry point, but was not specified as such in a SET,BREAKPOINT command.
*WARN - DATA WILL BE MOVED WITHOUT EDITING	Although the receiving data field in a COBOL CID MOVE command is alphabetic edited, alphanumeric edited, or numeric edited, the source field will be moved to the receiving field without editing.
*WARN - DESTINATION IS GROUP OR EDITED	The receiving item in a COBOL CID MOVE command is a group or edited item. If you allow the data to be moved, no editing will take place.
*WARN - EXISTING AUXILIARY FILE WILL BE CLOSED	A SET,AUXILIARY command has been issued which specifies a file name different from that of the existing auxiliary file.
*WARN - EXISTING BREAKPOINT WILL BE REDEFINED	An attempt is being made to set a breakpoint where one already exists. A positive acknowledgment causes the new definition to override the old one.
*WARN - EXISTING SUSPEND FILE WILL BE OVERWRITTEN	A SUSPEND command has been issued while a suspend file exists. The existing file could be due to a SUSPEND command issued earlier in the terminal session, prior to the current Debug session.
*WARN - FORMAL PARAMETER LENGTH ERROR	Character length of the formal parameter differs from that of the symbol table.
*WARN - GROUP xxx WILL BE REDEFINED	The name supplied in a SET,GROUP command is that of a currently existing GROUP. A positive acknowledgment causes the new definition to override the old one.
*WARN - LINE n NOT EXECUTABLE - LINE m WILL BE USED	The specified line number is not executable or is nonexistent. A positive acknowledgment causes line m to be used instead.
*WARN - MOVE TO GROUP LEVEL ITEM	A COBOL CID MOVE command specified the data name of a group item as the receiving field. A move to a group item does not cause conversion of the source data to the usages of the elementary data items within the group level receiving field.
*WARN - PERMANENT SUSPEND FILE WILL BE RETURNED	A SUSPEND command has been issued while a permanent suspend file exists without write or modify access.
*WARN - PROGRAM xxx HAS n ENTRIES - yyy WILL BE TRACED	The TRACEBACK command processor issues this warning when more than one entry point is encountered in a program module being traced. A positive acknowledgment results in a continuation of the traceback at yyy.
*WARN - STRING OVERFLOW - STRING WILL BE TRUNCATED	You are attempting to assign a string longer than 131070 6-bit characters to a BASIC program variable. A positive response causes a 131070 character string to be assigned.
*WARN - SUBSCRIPT OUT OF RANGE	A subscript for a FORTRAN variable in an IF, PRINT, or assignment statement is out of range. If a positive response is entered, the subscript value is accepted, resulting in the use of a location beyond that defined for the variable.
*WARN - TRAP #n, type, qualifiers, WILL BE CLEARED	A pending SET,TRAP command has a scope which overlays the scope of an existing trap of the same type. A positive acknowledgment CLEARS trap #n.
*WARN - WRITING INTO RA+1	ENTER or MOVE command, if processed, will write into location RA+1.

TABLE B-3. INFORMATIVE MESSAGES

Message	Significance
ASCII MODE MISMATCH BETWEEN PROGRAM AND I/O DEVICE	Either the BASIC program is declared to be in ASCII mode and the I/O device is not in ASCII, or the I/O device is in ASCII and the program is not. The program can continue to be debugged, but the data input and output by CID may not be what you expect. This message is always issued when debugging ASCII programs under NOS/BE, since CID does not support ASCII mode input and output under NOS/BE.
COMMAND LINE TOO LONG - EXCESS CHARACTERS DISCARDED	The command line contains more than 150 6-bit characters or 75 12-bit escape code ASCII characters. A syntax error will result if part of a command was discarded.
CYBER INTERACTIVE DEBUG	After the program to be debugged has been loaded, this message is issued when CID receives control. An initial set of traps and/or breakpoints should be established at this point before starting program execution.
CYBER INTERACTIVE DEBUG RESUMED	A debug session has been resumed from the point where it was suspended. The system command statement DEBUG (RESUME) has been entered following the issue of a SUSPEND command.
DEBUG ABORTED	This message is issued in response to QUIT,ABORT; it appears in the dayfile as well.
DEBUG SUSPENDED	This message is issued in response to a SUSPEND command; it appears in the dayfile as well.
DEBUG TERMINATED	This message is issued in response to QUIT or QUIT,NORMAL; it appears in the dayfile as well.
END COLLECT	Sufficient right brackets have been encountered to reduce the collect level to zero, thus ending collect mode. Interactive command mode is resumed; entered commands are immediately executed.
IN COLLECT MODE (,LEVEL n)	This message occurs when you receive CID control in collect mode; this was not the case when you last had control. Any subsequent commands entered will no longer be executed immediately, but will be checked for syntax and collected into a body or group for future execution. Level n is included in the message if a nested collect is in effect (n is greater than one). To end collect mode, n right brackets are required.
INTERRUPT IGNORED	CID was already in interactive command mode when a terminal interrupt occurred. Since the purpose of a terminal interrupt is to place CID in interactive mode, the interrupt is ignored.
INTERRUPTED	A terminal interrupt has occurred while a command sequence or a command which takes a list as a parameter was executing.
INTERPRET MODE TURNED OFF	As a result of clearing one or more traps, no traps remain that require interpret mode to be on. Subsequent program execution will be by direct execution of the machine instructions.
INTERPRET MODE TURNED ON	A SET,TRAP command has been issued with a trap type that requires interpret mode of program execution, and, currently, interpret mode is off. Subsequent program execution will be by interpreting all machine instructions.
INVALID x TREATED AS ;	A left bracket was found following a command other than SET,TRAP, SET,BREAKPOINT, or SET,GROUP; or a right bracket was found after a statement while not in collect mode.
LAST RESPONSE LINE IS DISCARDED	Command keyed-in as response to an error, warning, or interrupt prompt has issued an error or warning prompt. However, the total number of pending command sequences is equal to the highest number permitted and, therefore, the last response line has been discarded.

TABLE B-3. INFORMATIVE MESSAGES (Contd)

Message	Significance
NO BREAKPOINT xxxx	A request has been made to LIST, CLEAR, or SAVE a breakpoint at location xxxx. No such breakpoint exists. Any remaining list elements are processed.
NO BREAKPOINT #n	A request has been made to LIST, CLEAR, or SAVE a breakpoint #n which does not exist. Any remaining list elements are processed.
NO BREAKPOINTS	There are no breakpoints to LIST, CLEAR, or SAVE.
NO GROUP xxxx	A request has been made to LIST, CLEAR, or SAVE a group xxxx which does not exist. Any remaining list elements are processed.
NO GROUP #n	A request has been made to LIST, CLEAR, or SAVE a group #n which does not exist. Any remaining list elements are processed.
NO GROUPS	There are no groups to LIST, CLEAR, or SAVE.
NO SYMBOLS FOR OVERLAY (m,n)	A list element in a LIST,VALUES command specified an overlay that does not contain any user variables.
NO SYMBOLS FOR P.xxxx	A list element in a LIST,VALUES command specified a program module that does not contain any user variables.
NO xxxx TRAP yyyy	A request has been made to LIST, CLEAR, or SAVE a user-defined trap of type xxxx with scope yyyy. No such trap exists. Any remaining list elements are processed.
NO TRAP #n	A request has been made to LIST, CLEAR, or SAVE a user-defined trap #n which does not exist. Any remaining list elements are processed.
NO TRAPS	There are no trap definitions to LIST, CLEAR, or SAVE. Note that the three default traps, END, ABORT, and INTERRUPT, are never listed, cleared, or saved.
OVERLAY (m,n) NOT LOADED	A list element of a LIST,VALUES command has specified an overlay which is not currently loaded. Any remaining list elements of the LIST,VALUES command are processed.
PAUSE IGNORED FROM TERMINAL	This message results from entering PAUSE while in interactive (non-collect) mode.
PROGRAM xxxx NOT LOADED	A list element in a LIST,VALUES command specified a program module that has not been loaded. Any remaining list elements are processed.
TIME LIMIT	A time limit interrupt has occurred while either the program or a command sequence was executing. A small amount of time is left, sufficient to do a SAVE * and QUIT. To continue the session, enter SUSPEND followed by DEBUG(RESUME).
TRAP NUMBER IGNORED IN THIS CONTEXT	A trap number has been specified as a list element in a LIST,TRAP, CLEAR,TRAP, or SAVE,TRAP command of a form for which trap numbers are not allowed. Any remaining list elements are processed.
USER PROGRAM INTERRUPT PENDING	A terminal interrupt was detected while executing a high level language program compiled for use with CID. However, control was given to you first for some other reason besides result of the interrupt, such as a breakpoint or another type of trap. The pending interrupt will be acknowledged with an interrupt trap when the program execution is resumed.
USER RECOVER ROUTINE COMPLETED, x REQUESTED	CID issues this message after the program has completed its recover routine by making an ABORT or ENDRUN request.
VARIABLE NAMES NOT AVAILABLE FOR xxx	Either program xxx is not a high level program, or it was not compiled with the DB option explicitly specified or implicitly specified (by debug mode being on).

GLOSSARY C

Abort -
To terminate a program or job when a condition (hardware or software) exists from which the program or computer cannot recover.

Breakpoint -
A designated location in a program, where, if reached during program execution, a suspension of program execution occurs.

Common Block -
A module intended solely for storing data. As an alternative to passing data to routines via parameter values, a block of data can be declared in common to both the calling routine and the called routine.

Entry Point -
A location within a program that can be referenced from other programs. Each entry point has a unique name with which it is associated.

Extended Memory -
An auxiliary storage unit capable of high speed transfer to and from central memory. Refers to units formerly known as Extended Core Storage (ECS) and Large Central Memory (LCM).

Interactive -
Job processing in which you and the system communicate with each other, rather than processing in which you submit a job and receive output later.

Interpret -
The execution of computer machine instructions by other than direct means. A special routine called an interpreter examines each instruction to be executed and simulates its execution by the execution of several of its own instructions. Execution in interpret mode consequently takes 20 to 50 times as long as direct execution.

Interrupt (Verb) -
To stop a running program in such a way that it can be resumed at a later time.

Interrupt (Noun) -
A control signal that you issue from the terminal. If your program is executing when CID detects an interrupt, an INTERRUPT trap occurs; if a CID command sequence is executing, the command sequence is suspended and you gain control.

On NOS, CID interprets both the user-break-1 and the user-break-2 terminal keys as the interrupt key. The user-break-1 and user-break-2 keys differ, depending on the terminal type (see the Network Products Interactive Facility reference manual). On most terminals, these keys are CONTROL P and CONTROL T, respectively; you can issue an interrupt by pressing CONTROL P (or CONTROL T) followed by a carriage return.

On NOS/BE, you can issue an interrupt by pressing either %S followed by a carriage return or %A followed by a carriage return. (See the INTERCOM reference manual.)

Module -
A named section of coding or data. Prior to being loaded to form part of a program, modules are called object modules; after being loaded, they are called load modules.

Overlay -
A portion of a program, consisting of one or more modules, which can share an allocated area of memory with others of its kind. When access to a particular module is required, the overlay containing that module is loaded, thus overlaying the previous contents of the memory area allocated for that overlay.

Programs organized into overlays execute in an overlay environment. Such a scheme allows large programs to execute in a limited amount of memory.

Program -
The completely loaded set of one or more object modules. Such a set, where at least one of the modules is user-written, constitutes a program suitable for CID.

Program Module -
A module intended for program execution. A program module always has an entry point, a named location in the module to be used in calling the module via the RJ instruction.

Trap (Verb) -
The automatic transfer of control to a predefined location upon the detection of some specified condition.

Trap (Noun) -
The established mechanism for detecting a specified condition and causing a transfer of control. As implemented in CID, the special location to which control is transferred is in CID itself.

PROGRAM STRUCTURE D

This section describes where the CID executive module and your program are loaded in the field length and which files are accessed by CID for internal use.

PROGRAM LOAD

When a program is loaded, the loader reads the program into memory and allocates addresses to all of the program's constituent modules. For programs not organized into overlays, consecutively loaded modules are allocated consecutive ascending address areas.

CID can be used with programs having overlays if one of the following is true:

The FORTRAN routine OVERLAY loads the overlays.

or

The entry point DEBUG.OV is called whenever an overlay is loaded.

If a program is structured into overlays, that structure has the following organization. The first overlay in the program must be designated as the main (root or zero) overlay. It is allocated the lowest location in the program area of the field length and is always present in memory. This overlay is followed by the primary overlays. Following each primary overlay are secondary overlays, which are loaded beyond the end of their corresponding primary overlays.

A load map can be produced, if desired, by including a MAP,ON statement prior to calling for the load and execution of the program to be debugged.

Alternatively, the CID LIST,MAP command can produce load map information at any time during a debug session. This command is described in section 5.

A load map lists each module name and the actual starting address where each module is loaded. Entry point names and their addresses can also be listed. Each address given is the relative address from the start of the user field length. A load map also indicates if any modules are loaded in extended memory.

The map obtained from the loader is different according to whether debug mode is on or off. Two additional modules are included in the load of the program to be debugged when the program is loaded with debug mode turned on. One module, named DBUG., is allocated as the first module in the program load. DBUG. is located in the object library DBUGLIB. The other module, named UCLOAD, is allocated as the last module in the load. Both modules are loaded automatically when a relocatable object program is loaded with debug mode turned on. The minimum field length requirement for execution of your program is increased by the space required for these two modules. Figure D-1 shows the arrangement of code in the field length.

The presence of DBUG. in the load immediately following the job communication area increases the relative address at which each user module is loaded by the size of the DBUG. module.

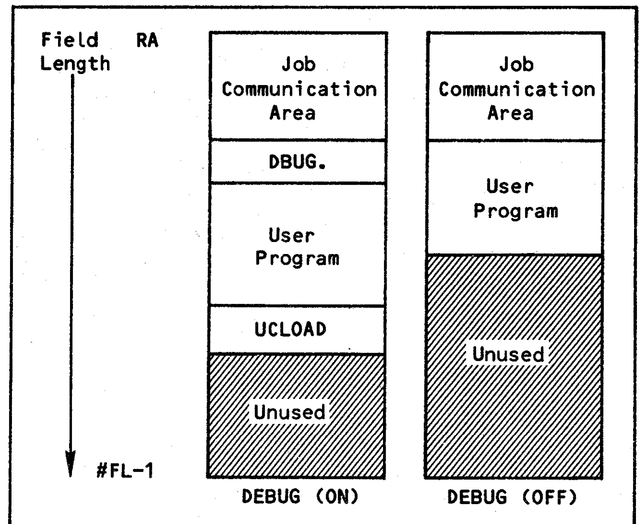


Figure D-1. Loader Field Length Assignment

BATCH CID FEATURES

CYBER Interactive Debug is primarily intended to be used interactively, but can be used in batch. In batch mode, you must place CID commands as the first system logical record in the file DBUGIN.

Output from CID is always written to a file called DBUGOUT. The type of output written to this file is controlled by the same command as is used to control output options to the terminal when CID is used interactively. (See the SET,OUTPUT command.) The default options for the standard output file in batch mode are E, W, D, I, R, B, and T. CID output can also be written to an auxiliary output file. (See the SET,AUXILIARY command.)

Error processing in batch mode is handled in the following manner:

- Commands in error are diagnosed and skipped. The next command from the input stream is used in place of the command in error.
- Commands that would normally give rise to warning messages are accepted, as if an ACCEPT response had been given in interactive mode.
- Veto mode is not effective in batch. The command to establish veto mode will be considered illegal in batch.

If the stream of commands on file DBUGIN is exhausted and the CID attempts to read another command, CID will behave as if the QUIT command had been executed. (See the QUIT command.)

Commands such as PAUSE or SET,BREAKPOINT without a body that would normally take input from the terminal take it from DBUGIN in batch. In order to use this feature, the commands on the DBUGIN file must be carefully ordered with this in mind; it is almost necessary to know in advance how the CID run will go. However, as mentioned above, DBUGIN is not accessed for responses in the error, warning, and veto cases.

Note that the position of DBUGIN is preserved across a SUSPEND/RESUME break.

USAGE CONSTRAINTS AND DEPENDENCIES F

The following CID usage constraints exist:

- Since part of CID resides in the field length below the lowest user routine, your program must not store into the region between the upper end of the job communication area and the FWA of the lowest user routine. CID is not protected from such damage, but it will prevent the DISPLAY, ENTER, and MOVE commands from being used to access this region.
- You should not use self-modifying code since this could overwrite breakpoint RJ instructions planted in your program by CID. Because FORTRAN generates self-modifying code, setting of breakpoints in FORTRAN programs at locations other than line numbers should be avoided.
- CID cannot be used to debug programs loaded by the segment loader.
- CID cannot be used to debug programs that use overlay capsules (OVCAPS).
- CID cannot be used to debug COBOL programs that:

  Have dynamically loaded subprograms

  Have fixed overlayable or independent segments

  Use the Message Control System

  Use the CYBER Database Control System (CDCS)
- CID cannot symbolically debug programs that have been loaded due to user call loading. Note that it is possible to reference locations in user call loaded programs by using their absolute addresses.
- CID cannot be used when loader debug facilities (for instance, TRAPPER) are being used.
- CID can only debug the most recently loaded program. That is, it is not possible to go through the sequence (LOAD(A), NOGO(ABSA), LOAD(B), NOGO(ABSB), ABSA. The loader only creates one file of tables for CID for each LOAD. Thus, if the sequence just given were executed, you would be trying to debug program A using the tables of program B.

  NOTE

  This limitation can be circumvented by saving file ZZZZZDT, the CID tables file, after NOGO(ABSA) and replacing it before ABSA is executed.
- Under NOS/BE, EDITLIB requires entry point names in a library to be unique. When overlays are generated with debug mode on, the entry point DBUG= is included in the (0,0) overlay. Thus, only one set of overlays generated in debug mode can be included in a library. This problem does not occur with relocatable routines, because CID is included at load time. (Under NOS, overlays cannot be stored in libraries.)
- Symbolic names that contain special characters must be enclosed in dollar signs ($) when the names are entered in language-independent commands, except as follows:

  BASIC source language symbols that contain dollar signs ($) do not have to be enclosed in dollar signs.

  COBOL source language symbols that contain hyphens (-) do not have to be enclosed in dollar signs.
- You must specify only the first seven characters of COBOL program module names in CID commands.
- CID cannot be used to debug programs that establish a long term connect with a system control point.
- The multiple input file connect feature available under NOS/BE cannot be used with CID.
- The stacked input feature, whereby you can type successive input lines before the input prompt is received, should be avoided. If an error or warning occurs, the pending stacked input will be used as your response.
- CID cannot be used to debug programs that directly call RPV to request error reprieve processing. Such usage would conflict with CID procedure. Instead, programs should call the COMPASS RECOVR macro or the FORTRAN RECOVR subroutine to specify error reprieve processing. CID contains special versions of these routines that provide proper interfacing to CID. (BASIC programs call the FORTRAN RECOVR subroutine when an ON ERROR condition occurs. COBOL programs do not use error reprieve processing.)
- CID cannot display, list, or print dynamically created arrays in BASIC; the array names are not recognized by CID unless declared with a DIM statement.

- Buffer flushing of the terminal output buffer of your program can only be accomplished if CID knows about that file. Compiler produced programs place entries in a list of files. A COMPASS program must be written to do this as well, or else CID will have no way of knowing such a file exists. When CID gets control from the program being debugged (for example, after a breakpoint occurs), it attempts to flush the prime terminal output buffer of the program, if it needs to be flushed. CID will search the file list, if any, established with the SETLOF macro, or the file list starting at address RA+2 for the first FET which indicates that it is writing on a terminal type device. If such a FET is found and if that FET indicates there is data in its buffer, CID will flush the buffer before interacting with you. This will ensure that output generated by the program before the break occurs is actually seen before the break occurs.

DEBUGGING HINTS G

The following suggestions can help you make more efficient use of CID:

- The central memory requirements are increased to approximately 55000_8 words when CID is used with programs smaller than 51000_8 words. When CID is used with programs larger than 51000_8 words, central memory requirements are increased by approximately 4000_8 words.

- In most cases, command sequences should be kept quite simple. With complex command sequences, a danger exists that more time will be spent debugging the command sequences than in debugging your program.

- Issue an appropriate SAVE command each time input collect mode is ended, rather than sometime later in the debug session. This way, if a definition is mistakenly cleared, the definition can be reestablished by reading the file, thus saving the need to reenter it from the terminal.

- Files created by the SAVE command are local files. It is your responsibility to make these files permament when desired for a future debug session.

- The auxiliary output file will not be saved or disposed to the printer by CID after a QUIT command. It is your responsibility to save or print this file when desired.

- Execution in interpret mode uses 20 to 50 times more computer time than direct execution. This fact should not discourage you from using interpret mode at all, since some of CID's most powerful features require interpret mode of execution. One way to keep run timings down to an acceptable level is to turn interpret mode off within program areas that have already been debugged, especially those areas which account for most of the execution time.

- If an abort trap occurs with #CPUERR equal to 1, 3, or 7, the reason could be an attempt to reference a location that was an unresolved external reference.

- When referencing high level program variables in the home program, use language-dependent CID commands (described in section 6) whenever possible.

  When the command references a debug variable or a symbolic address, language-dependent commands cannot be made.

- In FORTRAN programs, a frequently occurring bug is to pass too few parameters on a subroutine call. This can be caught by issuing both a STORE trap and a FETCH trap on location zero. The trap will occur when the first reference is made within the subroutine to the formal variable corresponding to the first missing parameter on the call.

- The SUSPEND/RESUME facility can be used to provide a checkpoint/restart capability. This capability allows you to save the debug session at various times, and if program data areas are damaged by program errors or CID commands, you can return to the session as it was previously saved.

  To save the session, proceed as follows:

  1. Enter the SUSPEND command with no parameters.
  2. Copy file ZZZZZDS to another file.
  3. Enter the DEBUG (RESUME) control statement.

  To return to the session as it was saved on another file, perform the following sequence of steps:

  1. Enter the QUIT command.
  2. Enter the DEBUG(RESUME,lfn) control statement, specifying as lfn the file to which you copied ZZZZZDS.

  Note that repositioning or closing of your files will not take place when you perform these sequences.

- The reading of program input files can be simulated with a suitable command sequence that is invoked in place of the read. The sequence should place distinct data values in those locations normally updated by the read routine.

  This technique eliminates the necessity of preparing test data input files, and solves the problem of user file positioning when using the checkpoint/restart technique outlined above.

  This technique also allows test data changes to be made during the debug session simply by issuing appropriate CID commands.

- It is not possible to chain to another program when CID is in control in the BASIC subsystem. When a CHAIN statement is encountered in a BASIC program, an END trap occurs.

  In order to use the CHAIN statement, use the QUIT command on the current debug session and manually issue the appropriate system commands to compile and execute the new program. In NOS, the chained-to program will already have been made the primary file.

- Avoid using the language-independent ENTER, MOVE, and DISPLAY commands with COBOL data items and BASIC string variables. These types of data are not word-aligned.

INDEX

Abort (definition) C-1
ABORT information variables 4-6
ABORT trap 3-2, 3-4
Absolute addresses 4-1
Accessing CID 2-1
Addition operator 4-8
Address range 4-4
Addresses 4-1
Assignment command (FORTRAN) 6-5
Automatic execution of CID commands 3-7
Auxiliary output file 2-2, 5-1, 5-8

BASIC features
    BASIC CID commands 6-1
    BASIC symbols 4-3
    Compilation 2-1
    LINE trap 3-3
    LIST,VALUES command 5-5
    STEP command 5-18
    #LINE debug variable 4-5
Batch CID features E-1
Blank (or unlabeled) common block 4-2, 4-4
Block referencing 4-4
Bodies 3-7
Breakpoint
    CLEAR,BREAKPOINT command 5-1
    Command sequence bodies 3-7
    Description 3-1, C-1
    LIST,BREAKPOINT command 5-3
    SAVE,BREAKPOINT command 5-7
    SET,BREAKPOINT command 5-9

Central memory field length D-1
Character sets A-1
CID concepts 3-1
CID features 1-1
CLEAR commands 5-1
CLEAR,AUXILIARY command 5-1
CLEAR,BREAKPOINT command 5-1
CLEAR,GROUP command 5-1
CLEAR,INTERPRET command 5-2
CLEAR,OUTPUT command 5-2
CLEAR,TRAP command 5-2
CLEAR,VETO command 5-2
COBOL features
    COBOL CID commands 6-2
    COBOL symbols 4-3
    Compilation 2-1
    LINE trap 3-3
    LIST,VALUES command 5-5
    PROCEDURE trap 3-4
    STEP command 5-18
    #LINE debug variable 4-5
    #PROC debug variable 4-5
Collect mode 3-7
Command sequences 3-7
Command syntax 4-1, 6-1
Command types 4-1
Comment 4-1
Common Block 4-2, 4-4, C-1
Compilation 2-1

Conditional commands
    BASIC IF command 6-1
    FORTRAN IF command 6-6
    Language-independent SKIPIF command 5-17
Constraints F-1

DBUG. module D-1
DB=ID compiler option 2-1
DEBUG control statement 2-1
Debug mode
    Compilation 2-1
    Execution 2-2
Debug Session 1-1, 8-1
Debug state variables 4-5
Debug user variables 4-5
Debug variables 4-5
Debugging hints G-1
Default traps 3-4
Dependencies F-1
Diagnostics 7-1, B-1
DISPLAY command
    COBOL 6-3
    Language-independent 5-12

ECS/LCM addresses 4-1, 4-4
Editing a command sequence 3-7
Ellipsis 4-4
END trap 3-2, 3-4
ENTER command 5-13
Entry points 4-2, C-1
Error
    Messages B-1
    Processing 7-1
    Responses 7-2
Example debug sessions 8-1
EXECUTE command 5-14
Execution address range 5-11
Execution under CID control 2-2
Expressions 4-6
Extended memory
    Addresses 4-1, 4-4
    Definition C-1

Features 1-1
FETCH trap 3-3
Files
    CID environment 2-3, 5-7
    Command Sequence 3-7
    Input 2-2
    Output
        Auxiliary output file 2-2, 5-1, 5-8
        Standard output file 2-2, 5-2, 5-10
    Scratch files 2-3
    Suspend file 2-3
FORTRAN features
    Compilation 2-2
    FORTRAN CID commands 6-5
    FORTRAN symbols 4-3
    LINE trap 3-3
    LIST,VALUES command 5-5
    STEP command 5-18
    #LINE debug variable 4-5

Glossary C-1
GO commands
    BASIC GOTO 6-1
    COBOL GO TO 6-3
    FORTRAN GOTO 6-6
    Language-independent GO 5-14
Groups
    CLEAR,GROUP command 5-1
    Description 3-7
    LIST,GROUP command 5-3
    READ command 5-17
    SAVE,GROUP command 5-7
    SET,GROUP command 5-9

HELP command 5-14
High level language features
    Compilation 2-1
    Debug variables 4-5
    Language-dependent commands 6-1
    LIST,VALUES command 5-5
    Source language symbols 4-3
    STEP command 5-18
    Traps 3-3, 3-4
Hints G-1
Home program
    Addresses 4-2
    Designation when execution is suspended 3-1
    SET,HOME command 5-9

Identifiers 4-3
IF command
    BASIC 6-1
    FORTRAN 6-6
Indirect addressing 4-9
Informative messages B-1
INSTRUCTION trap 3-3
Integers 4-4
Interactive C-1
Interpret C-1
Interpret mode
    CLEAR,INTERPRET command 5-2
    Description 3-6
    SET,INTERPRET command 5-10
    Variables 4-6
Interrupt
    Definition C-1
    INTERRUPT trap 3-3, 3-4
    Processing 7-1
    Responses 7-2
Introduction 1-1

JUMP command 5-15
JUMP trap 3-3

LABEL command 5-15
Labeled common block 4-2, 4-4
Language-dependent commands 6-1
Language-independent commands
    Descriptions 5-1
    Syntax 4-1
LET command 6-2
Line numbers 4-3
Line sequences 3-7
LINE trap 3-3
LIST commands 5-3
LIST,BREAKPOINT command 5-3
LIST,GROUP command 5-3
LIST,MAP command 5-5
LIST,STATUS command 5-5
LIST,TRAP command 5-5
LIST,VALUES command 5-5
Load map 5-5, D-1
Loader field length assignment D-1
Loading programs D-1
Locations 4-1

MAT PRINT command 6-2
MESSAGE command 5-15
Module 4-2, 4-4, C-1
Module referencing 4-2, 4-4
Module relative addresses 4-2
MOVE command
    COBOL 6-4
    Language-independent 5-15

Negative keywords 7-2
Notations xi
NULL command 5-16, 7-2
Numeric constants 4-4

Octal constants 4-5
Operators 4-8
Output options 5-9, 5-11
Overlay
    Addresses 4-2
    Definition C-1
OVERLAY trap 3-3

PAUSE command 5-16
Positive keywords 7-2
PRINT command
    BASIC 6-2
    FORTRAN 6-6
Procedure names 4-3
PROCEDURE trap 3-4
Program C-1
Program load D-1
Program module C-5
Program state variables 4-5
Program structure D-1

QUIT command 5-17

READ command 3-7, 5-17
Register state variables 4-6
Relative addresses 4-2
Report level 3-2, 5-11
Reprieve code 3-2, F-1
Response keywords 7-2
RJ trap 3-4

Sample debug sessions 8-1
SAVE commands 5-7
SAVE,BREAKPOINT command 5-7
SAVE,GROUP command 5-7
SAVE,TRAP command 5-7
SAVE,* command 5-7
Sequences 3-7

SET commands
    COBOL 6-4
    Language-independent 5-7
SET,AUXILIARY command 5-8
SET,BREAKPOINT command
    Collect mode 3-7
    Description 5-9
SET,GROUP command
    Collect Mode 3-7
    Description 5-9
SET,HOME command 5-9
SET,INTERPRET command 5-10
SET,OUTPUT command 5-10
SET,TRAP command
    Collect mode 3-7
    Description 5-11
SET,VETO command 5-12
SKIPIF command 5-17
Source language symbols 4-3
Statement labels 4-3
STEP command 5-18
STORE trap 3-4
Subtraction operator 4-8
SUSPEND command 5-19
Suspend/resume capability 2-1, 3-7, 5-19

TIME LIMIT 3-2
TRACEBACK command 5-19
Trap
    Action 3-4
    CLEAR,TRAP command 5-2
    Command sequence bodies 3-7
    Description 3-2, C-1
    LIST,TRAP command 5-5
    SAVE,TRAP command 5-7
    SET,TRAP command 5-11
    Types 3-2, 5-11

UCLOAD module D-1
Unlabeled (or blank) common block 4-2, 4-4
Usage constraints and dependencies F-1

Value operator (!) 4-8
Values 4-4
Variables
    BASIC 4-3
    Debug 4-5
    FORTRAN 4-4

Veto mode
    CLEAR,VETO command 5-2
    Definition 7-1
    Processing 7-1
    Responses 7-2
    SET,VETO command 5-12, 7-1

Warning
    Messages B-1
    Processing 7-1
    Responses 7-2

XJ trap 3-4

ZZZZZDS 2-1, 5-19, G-1
ZZZZZDT 2-3, F-1

#A debug variable 4-8
#B debug variable 4-8
#BP debug variable 4-5
#CPUERR debug variable 4-5
#EA debug variable 4-7
#ERRCODE debug variable 4-5
#EW debug variable 4-7
#FE debug variable 4-5
#FL debug variable 4-5
#GP debug variable 4-5
#HOME debug variable 4-5
#I debug variable 4-7
#INS debug variable 4-7
#INSL debug variable 4-7
#J debug variable 4-7
#K debug variable 4-7
#LINE debug variable 4-5
#OP debug variable 4-7
#P debug variable 4-5
#PA debug variable 4-7
#PARCEL debug variable 4-7
#PC debug variable 4-7
#PROC debug variable 4-5
#REG debug variable 4-8
#TP debug variable 4-5
#V1,...,#V10 debug variables 4-5
#X debug variable 4-8
_ Underline in addresses 4-2
! Value operator 4-8

COMMENT SHEET

MANUAL TITLE: CYBER Interactive Debug Version 1 Reference Manual

PUBLICATION NO.: 60481400

REVISION: D

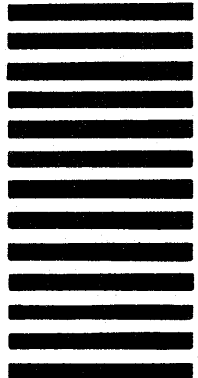
This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments on the back (please include page number references).

_____ Please reply _____ No reply necessary

FOLD FOLD

CUT ALONG LINE

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
P.O. BOX 3492
Sunnyvale, California 94088-3492

FOLD FOLD

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.
FOLD ON DOTTED LINES AND TAPE

NAME:

COMPANY:

STREET ADDRESS:

CITY/STATE/ZIP:

TAPE TAPE

BASIC CID COMMANDS

Command Format	Page
GOTO line-number	6-1
IF relexp THEN command	6-1
[LET] variable = expression	6-2
MAT PRINT array-list	6-2
PRINT output-list	6-2

COBOL CID COMMANDS

Command Format	Page
DISPLAY output-list	6-3
GO [TO] place	6-3
MOVE value TO data-item	6-4
SET name { TO value / UP BY amount / DOWN BY amount }	6-4

FORTRAN CID COMMANDS

Command Format	Page
variable = expression	6-5
GOTO statement-label	6-6
IF (logical expression) command	6-6
PRINT*,output-list	6-6

(Language-independent commands are shown on the inside front cover.)

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S

CONTROL DATA CORPORATION

102680323