Xerox ANS COBOL

Sigma 5-9 Computers

On-Line Debugger Reference Manual

90 30 60A

XEROX

Xerox ANS COBOL

Sigma 5-9 Computers

On-Line Debugger

Reference Manual

FIRST EDITION

90 30 60A

September 1973

File No.: 1X53
XQ22, Rev. 0

Printed in U.S.A.

NOTICE

This publication applies to the E00 version of the Xerox ANS COBOL compiler for BPM and CP-V.

RELATED PUBLICATIONS

Title	Publication No.
Xerox Sigma 5 Computer/Reference Manual	90 09 59
Xerox Sigma 6 Computer/Reference Manual	90 17 13
Xerox Sigma 7 Computer/Reference Manual	90 09 50
Xerox Sigma 8 Computer/Reference Manual	90 17 49
Xerox Sigma 9 Computer/Reference Manual	90 17 33
Xerox ANS COBOL (for BPM/CP-V)/LN Reference Manual	90 15 00
Xerox ANS COBOL (for BPM/CP-V)/OPS Reference Manual	90 15 01
Xerox Control Program-Five (CP-V)/TS Reference Manual	90 09 07
Xerox Control Program-Five (CP-V)/OPS Reference Manual	90 16 75
Xerox Control Program-Five (CP-V)/TS User's Guide	90 16 92
Xerox Control Program-Five (CP-V)/SM Reference Manual	90 16 74
Xerox Sort and Merge (for BPM/CP-V)/Reference Manual	90 11 99
Xerox Data Management System (DMS) (for BPM/CP-V)/Reference Manual	90 17 38
Xerox Extended Data Management System (EDMS)/Reference Manual	90 30 12

Manual Content Codes: BP – batch processing, LN – language, OPS – operations, RP – remote processing, RT – real-time, SM – system management, TS – time-sharing, UT – utilities.

CONTENTS

APPENDIXES

TABLES

COMMAND SYNTAX NOTATION

Notation conventions used in command specifications and examples throughout this manual are listed below.

Notation	Description
lowercase letters	Lowercase letters identify an element that must be replaced with a user-selected value. CRndd could be entered as CRA03.
CAPITAL LETTERS	Capital letters must be entered as shown for input, and will be printed as shown in output. DPndd means "enter DP followed by the values for ndd".
[]	An element inside brackets is optional. Several elements placed one under the other inside a pair of brackets means that the user may select any one or none of those elements. [KEYM] means the term "KEYM" may be entered.
{ }	Elements placed one under the other inside a pair of braces identify a required choice. {A / id} means that either the letter A or the value of id must be entered.
. . .	The horizontal ellipsis indicates that a previous bracketed element may be repeated, or that elements have been omitted. name[, name]. . . means that one or more name values may be entered, with a comma inserted between each name value.
⋮	The vertical ellipsis indicates that commands or instructions have been omitted. MASK2 DATA, 2 X'1EF' ⋮ BYTE DATA, 3 BA(L(59)) means that there are one or more statements omitted between the two DATA directives.
Numbers and special characters	Numbers that appear on the line (i.e., not subscripts), special symbols, and punctuation marks other than dotted lines, brackets, braces, and underlines appear as shown in output messages and must be entered as shown when input. (value) means that the proper value must be entered enclosed in parentheses; e.g., (234).
Subscripts	Sybscripts indicate a first, second, etc., representation of a parameter that has a different value for each occurrence. $sysid_1$, $sysid_2$, $sysid_3$ means that three successive values for sysid should be entered, separated by commas.

1. INTRODUCTION

The COBOL On-line Debugger is designed to be used with Xerox ANS COBOL, and operates under CP-V. The debugger is a special COBOL run-time library routine which is called by programs compiled in the TEST mode. This routine allows the programmer to monitor and control both the execution of his program and the contents of data-items during on-line execution. The debugger also allows the COBOL source program to be examined and modified.

The debugger can only be used during on-line execution; however, programs that have been compiled for use with the debugger may be run in batch mode. This is not recommended, though, because of the increased program size when the TEST mode is specified.

Summary of Debugging Capabilities

1. Setting breakpoints.
    a. Statement breakpoints.
    b. Procedure-name breakpoints.
    c. Data change breakpoints.
    d. Conditional breakpoints – statement, procedure, or data changes.
2. Displaying data.
3. Replacing data.
4. Branching.
5. Tracing statements as they are executed.
6. Tracing procedure-names as they are executed.
7. Displaying statement execution history.
8. Displaying procedure-name execution history.
9. Interrupting execution.
10. Resuming execution.
11. Executing single statement.
12. Executing single paragraph.
13. Dumping areas of program on line printer.
14. Displaying the status of user's files (FDs).
15. Executing a file of IASSIGN commands.
16. Removing break points.
17. Quitting the debugging run.
18. Displaying source lines.
19. Inserting source lines.

20. Replacing source lines.

21. Deleting source lines.

22. Displaying active breakpoints.

23. Defining abbreviations for data-names.

24. Typing a summary of debug commands.

25. Recovering from exception conditions – traps, I/O errors.

2. DEBUGGER INTERFACING

The debugger is the interface between the COBOL object program and the on-line user. The interface consists of (1) calls to the debugger that are added to the program when it is compiled in TEST mode; (2) a debug file that is produced during the TEST mode compilation; and (3) debug commands entered by the user.

Debugger Calls

When a program is compiled in TEST mode, calls to the debugger are generated prior to each procedure-statement and procedure-name. These calls are used to control the execution of statement and procedure-name tracing, breakpoints, and branching commands.

Debug File

The debug file is generated by the compiler, and contains a record for each data-name defined in the source program. These records contain the information which would be printed on the Data Map (base, displacement, size, record-qualifier), plus information on number of subscripts and the subscripting factors. The compiler allows for up to 255 occurrences of a data-name.

Debugger I/O

This section is included as an aid to understanding of how the debugger functions, and is of limited usefulness to the casual user.

The debugger does its I/O through the "monitor" DCBs listed below. These DCBs appear on a load map, and contribute to the size of the load module being debugged.

DCB	Use
M:BI	For reading REF/DEF stack of load module.
M:CI	For reading debug file.
M:DO	For output from DUMP command.
M:EI	For reading IASSIGN file used with SETFILES.
M:SI	For accessing user's source program.
M:UC	For communicating with the programmer; command input, data display, diagnostics.

3. DEBUGGER COMMAND LANGUAGE

This chapter covers general rules for forming debugger commands, special symbols for describing debugger commands, and definitions of the elements referred to in the commands. The actual syntax of the individual commands is given in Chapter 5.

General

Debugging commands are simple and readable, avoid artifical codes, and are intended to be COBOL-like to minimize confusion on the part of the programmer. The rules for names and operators are basically the same as for COBOL. Names must be composed of the alphabetic characters (excluding blank), the digits 0 through 9, and the hyphen (-), and must be preceded and followed by spaces. The allowable operators are EQ (or '='), LT (or '<'), and GT (or '>') and must also be surrounded by spaces.

The semicolon is used either as a command separator for attached commands or as a continuation indicator when it occurs as the last character before the end of a line (see the AT and WHEN commands).

Typographical Conventions Used in This Manual

Chapter 5 describes the various debugging commands and their specifications. The following conventions are used in explaining the format of the commands, and in examples.

1. Lowercase items indicate programmer-supplied data, to be replaced with actual names, literals, etc.
2. Capitalized items must be entered exactly as they appear.
3. All other symbols (except for brackets, braces, and elipses) are entered exactly as they appear.
4. Items enclosed in brackets [ ] are optional.
5. Items stacked inside braces { } indicate a choice must be made by the user.
6. Ellipses indicate repetition. For example,

   [;command]...

   means that a semicolon followed by a command is optional, and that more than one semicolon-command pair may occur.
7. The special symbol ⏎ is used for a carriage return, new line, or line feed. All commands must be terminated by ⏎, or ; ⏎.
8. Words and specifications are delimited by spaces. Words may also be terminated by commas, semicolons, colons, periods, and ⏎.

General Definitions

The following elements are referenced in debugger commands, and are explained here to avoid repetition of the explanations.

1. Identifier — an item that exists in the Data Division.
2. Literal — a self-defining symbol.

3. Location – the specification of a statement, or procedure-name, within the Procedure Division.

4. Breakpoint – not a command element, but a basic concept for using the debugger.

Detailed explanations of these elements are given below.

Identifiers

An identifier is a COBOL data-name together with any necessary qualifiers or subscripts. The debugger recognizes all data-names defined in the program except those that appear in the REPORT or LINKAGE sections. The debugger expects identifiers to conform to the basic COBOL rules for uniqueness of data reference (refer to XEROX ANS COBOL Reference Manual 90 15 00). In addition, an identifier may be qualified by the program-id of the program in which it occurs. In this case, the program-id must precede the subscripts (if any) associated with the identifier, and must be prefixed with a colon. No space is required before the colon.

The debugger does no range-check on subscripts, nor does it check for the occurrence of the maximum number of subscripts. If fewer subscripts are given than required, all unspecified subscripts will be assumed to be equal to 1. For instance, if three subscripts were required for a data name and none were given, a subscript of (1, 1, 1) would be assumed.

Literals

The debugger recognizes three types of literals: numeric, alphanumeric, and hexadecimal.

A numeric literal consists of a leading sign (+, -, or blank), and up to thirty numeric digits. No imbedded periods or commas are allowed. Numeric literals may be used as operands only in SET or IF commands where the other operand is display numeric, packed numeric, or integer binary. Numeric literals are always used as right-justified numbers with leading zeroes or truncation, as dictated by the picture of the other operand.

Alphanumeric literals are formed by enclosing any string of characters within single quotes. An alphanumeric literal may be entered on multiple lines by ending all lines except the final one with the sequence ; ⏎. In this case neither the semicolon nor the ⏎ will be included in the literal. If the programmer wants to include a ⏎ in the literal, he can do so by terminating a line with a ⏎, not preceded by a semicolon, and continuing on the next line. An alphanumeric literal must be terminated by a single quote mark. The maximum allowable size for alphanumeric literals is 250 characters. Alphanumeric literals are always treated as left-justified. If an alphanumeric string contains fewer characters than an identifier to which it is being moved or compared, blanks are added to the right of the literal to insure matching sizes.

Hexadecimal literals are formed by entering the character X, followed by a hexadecimal string. A hexadecimal string consists of a combination of the digits 0 through 9, and the letters A through F, enclosed in single quotes. Each digit or letter represents half of an 8-bit byte. The leftmost character of the hexadecimal string corresponds to the leftmost half-byte of the identifier to which the literal is related. If the literal is shorter than the identifier, the literal will be extended with zeros.

Locations

A location is a symbol that refers to a particular point in the Procedure Division of a program. A location may be either a procedure-name or a statement-identifier.

Procedure-names are the paragraph or section names used in the program. Paragraph names may be qualified by the section in which they appear, if necessary. All-numeric procedure-names must be preceded by the '#' character. Note: 101 is an all-numeric procedure-name, but 100-01 is not – the hyphen makes the difference.

Statement-identifiers are generated for each statement in the Procedure Division. Procedure-names are considered to be statements. Statement-identifiers have the format:

llll [.sss][(vv)]

where

llll is the sequential line-number assigned by the COBOL compiler

sss is the sub-line-number which is assigned to COPYed lines

vv is the statement number within the line.

Note that the first statement on a line is number zero, not number one.

Assuming that line 574 of the program is

PARA-37. IF A = B MOVE P TO Q.

then 574 refers to the paragraph name, 574(1) to the IF statement, and 574(2) to the MOVE statement.

Breakpoints

Breakpoints are those places in the program at which the programmer wishes to interrupt (or break) the normal execution flow of the program. Breakpoints may be used to display or alter data, perform branching to change the flow of the program, or to halt program execution so that the programmer can enter debugger commands.

There are three kinds of breakpoints – procedure breakpoints, in effect only when program execution reaches a particular location; data breakpoints, in effect any time the contents of an identifier are changed; and immediate breakpoints, in effect whenever the break key is depressed.

4. TYPICAL USE OF DEBUGGING COMMANDS

This chapter gives some simple examples of the usage of several of the more common debugger commands. Detailed descriptions of all the commands are given in Chapter 5, and Appendix B contains a complete sample debugging session. These command examples are intended to highlight some of the features of the debugger.

Commands may be entered whenever the debugger prompts with the > character. This will occur at the beginning of the session, whenever a STOP command is encountered at a breakpoint, and whenever an abort condition is detected (I/O error, trap, etc.).

In general, commands are followed by a carriage return, line feed, or new line character. The symbol for these characters is ⏎. In these examples each line is implicitly followed by ⏎ even though it is not shown.

The commands illustrated in this chapter are

SETFILES	PRINT	DROP	OFF	SOURCE
GO	PRINTX	AT	OFFP	
NEXT	SET	WHEN	OFFS	
NEXTP	EQUATE	IF	OFFWN	

SETFILES

This command is used to instruct the debugger to access a file of !ASSIGN commands, and use the information in it to control the assignment of the FDs in the program. The command,

```
SET FILES  PROGRAM-410-A-ASSIGNS
```

tells the debugger to assign the FDs as indicated in the file PROGRAM-410-ASSIGNS.

GO

This command instructs the debugger to begin (or resume) execution of the program. If a location is specified, execution begins at that location, otherwise execution begins at the current statement.

The command

```
GO  READ-A-RECORD
```

instructs the debugger to begin execution of the program at procedure-name READ-A-RECORD.

The command

```
GO  1024(2)
```

instructs the debugger to begin execution of the program with the third statement on line number 1024.

NEXT and NEXTP

These commands instruct the debugger to begin executing the program, but only to execute a single statement or procedure-name, respectively. As with the GO command, execution begins either at the referenced location, or the current statement.

The command

```
NEXT  READ-A-CARD
```

instructs the debugger to execute the statement at procedure-name READ-A-CARD. This will only result in a branch within the program, since the first statement of a procedure-name is just the procedure-name itself.

The command

```
NEXTP  792(1)
```

instructs the debugger to begin execution at statement two of line number 792, and continue executing until a procedure-name is encountered.

In all cases, the debugger gives a standard break message and prompts for more commands when the next statement or procedure-name is reached.

PRINT and PRINTX

These commands instruct the debugger to display the contents of an identifier. The PRINT command causes the data in the identifier to be printed as it is described in the COBOL program. The PRINTX command causes the data to be printed in hexadecimal notation.

The command

```
PRINT  EMPLOYEE-NUMBER
```

would cause the debugger to print the current value of the identifier EMPLOYEE-NUMBER, which might be 010308.

The command

```
PRINTX  EMPLOYEE-NUMBER
```

would cause the same data to be printed in hexadecimal, which would be F0F1F0F3F0F8.

SET

This command tells the debugger to change the contents of an identifier to that specified in the command.

The command

```
SET  SUBSCRIPT-VALUE = 3
```

will move the number 3 into SUBSCRIPT-VALUE.

EQUATE and DROP

These commands cause the debugger to establish and discard, respectively, an abbreviation for an identifier. This can save the programmer quite a few key-strokes and, consequently reduces the chances of error.

The commands

```
EQUATE SSNA SOCIAL-SECURITY-NUMBER IN RECORD-A

EQUATE SSNB SOCIAL-SECURITY-NUMBER IN RECORD-B
```

cause the debugger to put SSNA and SSNB in the abbreviation table, and the commands

```
DROP SSNA

DROP SSNB
```

cause SSNA and SSNB to be removed from the table.

AT and WHEN

These commands establish breakpoints. The AT command establishes a breakpoint at a particular location, and the WHEN command establishes a data-change breakpoint. Either command may be followed by other debugger commands which will be executed each time the breakpoint is reached. This allows the programmer to modify the execution of his program while he is debugging. If no commands follow the AT or WHEN command, the debugger automatically halts program execution and prompts for a command when the breakpoint is reached.

The commands

```
AT LOGIC-ERROR; PRINT FIELD-1; PRINT FIELD-2; STOP

AT 3784

AT SEARCH-TABLE-1; SET TABLE-1-SUBSCRIPT = 1

AT 1401; GO 1401(2)

WHEN TABLE-ENTRY (14)
```

have the following effects on execution of the program:

1. When the procedure-name LOGIC-ERROR is reached, the contents of FIELD-1 and FIELD-2 will be printed out, and the debugger will prompt for a command. When line 3784 is reached, execution will halt, and the debugger will prompt for a command.

2. When procedure-name SEARCH-TABLE-1 is reached, TABLE-SUBSCRIPT will be set to 1, and execution will continue.

3. When statement 1401 is reached, statements 1401 and 1401(1) will be skipped, with execution continuing with statement 1401(2).

4. When the contents of TABLE-ENTRY (14) are changed, execution will halt and the debugger will prompt for a command.

IF

This command provides the programmer with the capability of having the debugger make simple decisions for him. This is especially useful when the IF command follows an AT command. This allows him to direct the debugger to execute a series of commands only if a particular condition exists when a breakpoint is reached.

The command

```
AT 2741; IF NUMBER-OF-SAMPLES = 0; SET CALCULATED-AVERAGE = 0; GO CALCULATION-COMPLETED
```

tells the debugger to set the result of a series of calculations to zero, and skip the calculations whenever the operand NUMBER-OF-SAMPLES is zero before statement 2741 is executed.

OFF, OFFP, OFFS, and OFFWN

These commands are used to remove breakpoints. OFF and OFFWN are used to remove a single breakpoint set by the AT or WHEN commands, respectively. OFFP removes all breakpoints whose location is a procedure-name, and OFFS removes all breakpoints whose location is a statement-id.

The commands

```
OFFWN INPUT-MAJOR-CONTROL-FIELD

OFF COMPUTE-AVERAGES

OFF 7080(2)
```

cause the data-break on INPUT-MAJOR-CONTROL-FIELD, the procedure-name breakpoint at COMPUTE-AVERAGES, and the statement-id breakpoint at 7080(2) to be removed.

SOURCE

This command prints out a line or group of lines from the source program. This enables the programmer to quickly determine what fields are involved in a statement that causes a decimal trap, or some other error.

The command

```
SOURCE 7074,3
```

causes the debugger to print out lines 7074, 7075, and 7076.

5. DESCRIPTION OF COMMANDS

This chapter describes the syntax and scope of each of the debugger commands. The conventions and definitions given in Chapter 3 apply in all command descriptions in this chapter.

For ease of reference the commands are divided into groups, based on the type of function performed by the command. These groups are Breakpoint Control, Stepping and Branching, Data Display and Change, Source Manipulation and Display, Execution Trace, and Miscellaneous.

Breakpoint Control Commands

This group of commands contains the commands that establish and remove breakpoints, plus those that are of little or no use except when attached to breakpoints. The commands in this group are

AT	STOP	OFF	OFFP	LISTBRKS	RUN
WHEN	IF	OFFWN	OFFS	(BRK)	

AT

AT, the basic breakpoint command, establishes a breakpoint at a specific location in the program. The breakpoint is effective just before execution of the COBOL instructions at the referenced location. In the case where the referenced location is a procedure-name, the breakpoint is effective before any statements in the referenced procedure are executed. Debug commands may be specified as part of the AT command. These commands would be automatically executed each time the breakpoint is effective. If no additional debugger commands are specified for the breakpoint, the debugger will halt execution of the program when the breakpoint is reached.

The general form of an AT command is

AT location[;command]...

The optional commands may be any valid debugger commands. No syntax validation is done on these commands until the breakpoint is reached. If the optional command list exceeds a single input line, a semicolon followed by a (RET) entered at the end of a line continues it to the next. However, individual commands must be contained on a single line.

WHEN

The WHEN command establishes a breakpoint that is effective whenever the contents of an identifier are changed. The debugger will detect any change in the contents of the identifier, such as a move to a group item which contains the identifier, or a move to an identifier which redefines the identifier. A move to the identifier which does not change the value of the identifier – such as moving spaces to a field that is already blank – does not cause a breakpoint.

A series of commands may be appended to the WHEN command. These commands may be any valid debugger commands, and they will be executed whenever the breakpoint is effective. Command validation is not performed until the breakpoint is effective.

The general form of the WHEN command is

WHEN identifier[;command]...

If the optional command list exceeds a single input line, entering a semicolon followed by a (RET) at the end of the line continues it to the next. The next line of commands will then be treated as a continuation of the command list

to be executed at the time the data breakpoint associated with the WHEN command is effective. In the absence of a command list, the debugger will halt execution of the program when the breakpoint is effective.

STOP

The STOP command is used to halt execution of the program at a breakpoint established by an AT or WHEN command. The STOP command must be the final command in the command list for the breakpoint. If no command list is given for an AT or WHEN command, the debugger automatically inserts a STOP command.

The general form of a STOP command is

```
STOP
```

IF

The IF command is used to provide conditional execution of debugger commands at a breakpoint established by an AT or WHEN command. All commands which follow the IF command are ignored and execution continues unless the conditional expression in the IF command is true.

The general form of an IF command is

```
IF expression; command[;command]...
```

Valid expressions are of the form

```
identifier relation literal
```

The valid relations are equal to (EQ or =), less than (LT or <), greater than (GT or >), not equal (NE), not less (NL), and not greater (NG).

An IF command may follow another IF command; however, it must be understood that the second IF command will only be evaluated when the expression in the first IF command is true. The debugger has no capability for handling IF statement with an ELSE. ELSE NEXT SENTENCE is always implied.

OFF

The OFF command removes a breakpoint that has previously been established by an AT command. It also removes any command list associated with the breakpoint.

The general form of the OFF command is

```
OFF location
```

OFFWN

The OFFWN command removes a breakpoint that was established by a WHEN command. It also removes any command list associated with the WHEN command.

The general form of the OFFWN command is

```
OFFWN identifier
```

OFFS

The OFFS command removes all breakpoints that (1) were established with an AT command, and (2) have a statement-id as their location.

The general form of the OFFS command is

```
OFFS
```

OFFP

The OFFP command removes all breakpoints that (1) were established with an AT command, and (2) have a procedure-name as their location.

The general form of the OFFP command is

```
OFFP
```

LISTBRKS

The LISTBRKS command causes the debugger to print out all the breakpoints that are currently established. All breakpoints established by AT commands with procedure-names as their locations will be listed first, then those established by AT commands with statement-ids as their location will be listed, and finally all breakpoints established by WHEN commands will be listed. All breakpoints established by WHEN commands will be printed out preceded by the identification DATA BREAK. No command lists will be printed.

The general form of the LISTBRKS command is

```
LISTBRKS
```

(BRK) — The Break Key

The (BRK) command causes the debugger to establish a temporary breakpoint before the execution of the next COBOL statement. No command list can be established, however the debugger will halt program execution and prompt for a command.

Pressing the break key gives the break command.

RUN

The RUN command removes all breakpoints established by AT or WHEN commands, and continues execution of the program. If a location is specified in the RUN command, the debugger will resume execution of the program at that location. If no location is given, execution resumes with the next statement.

The general form of the RUN command is

```
RUN[location]
```

Stepping and Branching Commands

These commands provide a great deal of power and flexibility to the programmer by allowing him to step through the program, checking for correct operation as he proceeds and giving him the option of changing the execution flow of the program without recompiling.

The stepping and branching commands are

```
NEXT          NEXTP          GO
```

NEXT

The NEXT command causes the debugger to execute a single statement of the program. If a location is given in the NEXT command, the statement at the referenced location is executed, otherwise the next statement in the program is executed. Note that the debugger considers procedure-names to be statements, so that no actual program instructions will be executed if execution is currently stopped at a procedure-name, or if the location given in the NEXT command is a procedure-name.

The general form of the NEXT command is

```
NEXT[location]
```

NEXTP

The NEXTP command causes the debugger to resume execution of the program until the next procedure-name is encountered. If a location is given in the NEXTP command, execution resumes at the referenced location, otherwise execution proceeds with the next statement of the program. Note that if the location given in the NEXTP command is a procedure-name, or the statement-id of a procedure-name, no actual program instructions will be executed.

The general form of the NEXTP command is

```
NEXTP[location]
```

GO

The GO command causes the debugger to begin execution of the program. If no location is given, execution begins with the current statement of the program. If a location is given, control is transferred to that location before execution begins.

Data Display and Manipulation Commands

These commands are useful for checking for correct results from executing a portion of the program, and for correcting results that are not as expected. Since COBOL identifiers tend to be rather lengthy and difficult to key in, a facility for defining abbreviations is included in this group of commands. For those occasions when nothing more can be done on line, a high-volume data dump to the line printer (or a file) is also provided.

The commands in the group are

PRINT	SET	DROP
PRINTX	EQUATE	DUMP

PRINT

The PRINT command causes the contents of an identifier to be displayed on the terminal. The display is in the natural mode of the identifier. The types of identifiers and their natural modes are given in Appendix A.

The general form of the PRINT command is

```
PRINT identifier
```

PRINTX

The PRINTX command displays the contents of an identifier in hexadecimal mode. This is useful for detecting non-printing invalid characters in a field.

The general form of the PRINTX command is

```
PRINTX identifier
```

SET

The SET command places a specified value in an identifier. It can be used for initializing and correcting the contents of identifiers. The debugger does not provide decimal point alignment or accept decimal points in numeric literals. It will not accept a numeric literal as a value for a nonnumeric identifier. The debugger will detect and diagnose attempts to SET an identifier to a literal that exceeds the size of the identifier.

The general form of the SET command is

SET identifier = literal

EQUATE

The EQUATE command enters an abbreviation for an identifier into the abbreviation table. The debugger checks that the proposed abbreviation has not already been used, either as an abbreviation or a data-name. A diagnostic is issued if a duplicate is detected, and no abbreviation is entered.

The debugger determines the actual memory address of the identifier at the time the EQUATE command is issued. This means that if the identifier contains a subscript, the value of the subscript when the EQUATE is executed will determine where the debugger thinks the first occurrence of the identifier is in memory. In general, it is best to ignore the subscripts when defining the abbreviation, and to remember them when using the abbreviation.

Abbreviations are limited to seven characters, and may be formed from any combination of letters, digits, and the hyphen, except that an abbreviation cannot begin with a hyphen.

The general form of an EQUATE command is

EQUATE abbreviation [TO] identifier

DROP

The DROP command removes an abbreviation from the table.

The general form of the DROP command is

DROP abbreviation

DUMP

The DUMP command provides a hexadecimal dump through the M:DO DCB. This dump is in the standard format provided by the !PMD control card during batch runs. The specifications for the dump, including commentary, are written out as a separate line preceding the actual dump.

The DUMP command must be the last command if it is part of a command list associated with a breakpoint.

From-identifier and to-identifier define the lower and upper limits of the dump. If no to-identifier is given, the from-identifier will be dumped. If the to-identifier has an actual memory address which is lower than that of the from-identifier, only the contents of the from-identifier will be dumped.

In all cases, the dump will consist of whole words of core memory, and the programmer is cautioned that both the first and last words of the dump may contain data not contained in the from- or to-identifiers.

The general form of the DUMP command is

DUMP from-identifier TO to-identifier commentary (RET)

Execution Tracing Commands

This group of commands provides the user with the ability to trace the execution flow of the program at either the statement level or the procedure-name level, or both. Tracing can be done as the program executes, or a trace of the previous execution flow can be requested while program execution is halted.

The commands in this group are

STRACE SLIST PTRACE PLIST

STRACE

The STRACE command controls statement execution trace mode. When statement execution trace mode is on, the statement-id of each statement is printed out just before the statement is executed. Statement trace mode is normally off. If on, using the word OFF in the STRACE command turns the statement trace mode off.

The general form of the STRACE command is

STRACE [OFF]

SLIST

The SLIST command is used to display the history of statement execution. The display will consist of the statement-ids of the last n statements executed, in reverse order. That is, the current statement-id will be displayed first, then the one just previously executed, etc. If no value n is specified, the entire trace table will be displayed in forward order. The size of this table is given in Appendix A.

The general form of the SLIST command is

SLIST[,n]

PTRACE

The PTRACE command controls the procedure-name execution trace mode. When the procedure-trace mode is on, each procedure-name is printed out prior to execution of the procedure. Procedure-trace mode is normally off. If on, using the word OFF in the PTRACE command turns the procedure-trace mode off.

The general form of the PTRACE command is

PTRACE [OFF]

PLIST

The PLIST command lists the contents of the procedure-name execution history table. If the parameter n is given, the last n procedure-names in the table are listed in reverse order, i.e., most recently executed first. If no n is given, the entire table is listed, with the oldest entry first. The size of the table is given in Appendix A.

The general form of the PLIST command is

PLIST[,n]

Source Display and Miscellaneous Commands

The commands in this group enable the programmer to examine and modify his COBOL source program while debugging the program.

The commands in this group are

SOURCE REPLACE INSERT DELETE

SOURCE

The SOURCE command prints out one or more source lines on the programmer's terminal. This enables the programmer to look at the statement being executed when, for instance, a decimal data error occurs. From the source statement he can determine which data items might be causing the error. Then he can use the PRINT command to determine which data item is actually the problem, and he can correct it with the SET command.

The general form of the SOURCE command is

SOURCE line-number[,n]

where line-number is the line-number assigned by the compiler, and n is the number of lines to be printed. Note that source lines that are copied into the program do not exist as part of the source file, and cannot be printed.

REPLACE

The REPLACE command allows the programmer to replace an existing source statement. The replacement has no effect on the current debugging run. This enables the programmer to make a correction to a line at the time an error is discovered.

The general form of the REPLACE command is

REPLACE line-number

INSERT

The INSERT command allows the programmer to add new lines to the source file. The added lines have no effect on the current debugging run. If an attempt is made to INSERT a line that already exists, the debugger reports an error. A series of lines may be inserted by specifying an increment in the INSERT command. The debugger will then prompt for more lines until either a null line (@ only) is entered, or the addition of the increment to the line number of the last line entered would result in the next line not following the last line entered.

The general form of the INSERT command is

INSERT line-number[,increment]

DELETE

The DELETE command deletes a line from the source file.

The general form of the DELETE command is

DELETE line-number

Miscellaneous Commands

These commands do not fall conveniently into any of the other groups. The commands in this group are

SETFILES LISTFILE HELP QUALIFY END

SETFILES

The SETFILES command allows the programmer a quick and easy method for ensuring that all his DCBs are correctly assigned. It not only saves him the time and effort involved in keying in a series of !SET commands before he begins debugging a program, it also allows him to minimize any differences that exist between !ASSIGN and !SET commands.

The debugger analyzes the file specified by the SETFILES command, ignoring all commands except for !ASSIGNs. The !ASSIGN commands are analyzed, and the DCBs are assigned as specified.

The general form of the SETFILES command is

SETFILES filename

LISTFILE

The LISTFILE command displays the status of an FD in the program.

The general form of the LISTFILE command is

LISTFILE fd-name

HELP

The HELP command lists the debugger commands, with a short description of each.

The general form of the HELP command is

HELP

QUALIFY

The QUALIFY command is used to change the default value of program-id, the name of the source-file to be used in source manipulation commands, or the debug file to be used. This is only useful when the load module being executed is composed of two or more COBOL programs that were compiled with the TEST option.

The general form of the QUALIFY command is

QUALIFY [program-id][,[source-file][,debug-file]]

END

The END command terminates the debugging session, and returns control to the monitor.

Error/Abort Messages

Errors are detected by any or all of the following:

Debugger

COBOL run-time library routines

Monitor

When errors are detected by the monitor, it notifies either the debugger or the COBOL run-time library. When errors are detected by the run-time routines, they notify the debugger. The debugger always notifies the programmer and gives him a chance to continue debugging whenever it detects or is informed of an error by the monitor or the run-time routines.

Errors detected by the COBOL run-time routines are generally reported to the user before they are reported to the debugger. The debugger does not gain control until after the run-time routine has attempted to abort the job.

Errors detected by the debugger are confined to errors in command specifications. These errors are diagnosed as shown in Appendix B, and the debugger prompts for another command.

Errors detected by the monitor result in the monitor's passing control to the debugger. The debugger will then issue an appropriate diagnostic message, and prompt the user for another command.

Monitor-detected errors include decimal data exceptions, memory protect violations, and user aborts or end of job.

Run-time-detected errors include I/O errors and improper transfers of control.

When attempting to resume execution of the program, the user should remember that the debugger will resume execution with the statement that caused the error. Unless the cause of the error, say bad decimal data, has been corrected, this will result in another occurrence of the same error.

6. OPERATIONS

The COBOL On-line Debugger is designed for use with the CP-V monitor. This chapter explains the steps involved in compiling, loading, and debugging a COBOL program on-line.

The emphasis in this chapter is on what commands are used to accomplish various results, not on explaining all options of every command. More detailed explanations for all commands shown (except for !COBOL and debugger commands) can be found in the Xerox CP-V Time-sharing Reference Manual, 90 09 07.

Compiling

When compiling a COBOL program for use with the debugger, the programmer must be sure to do the following

1. Specify the TEST option on his COBOL control command. This directs the compiler to build a keyed source output file (SO) and a debug file for the program.

2. Use !ASSIGN or !SET commands to inform the compiler on what files to write the SO and debug files.

The compilation can be done either on-line, or in the batch mode. Any valid COBOL control command options may be used in addition to the TEST option. Either the GO or BO option must be used, with the appropriate DCB assignment, so that the compiler will create an object-module (ROM) for the program.

Loading

The compiled program, the COBOL run time library routines it requires, and the debugger must be combined to form an executable load module (LMN) before any debugging can proceed. Either the batch loader (!LOAD) or the on-line loader (LINK) can be used for this purpose. The batch loader must be used if the resulting load module is to be overlaid.

Whichever loader is used, to include the debugger, the loader is instructed to search the special library account CDBGLIB before searching the normal library account COBLIB in order to satisfy external references in the ROM. The option used to do this when using the batch loader is written as

```
(UNSAT,(CDBGLIB),(COBLIB))
```

When using the on-line loader, the libraries are referenced as

```
BLIB:.CDBGLIB,BLIB:.COBLIB.
```

An example of an on-line load command is given below.

```
!LINK MYROM,X23;BLIB:.CDBGLIB,BLIB:.COBLIB OVER MYLMN
```

This instructs the on-line loader to form a load module named MYLMN, which will replace any existing LMN with the same name. To build the LMN, the loader is instructed to use the object modules named MYROM and X23, and to satisfy external references that occur in these two ROMs by using the two on-line libraries, BLIB: in account CDBGLIB, and BLIB: in account COBLIB.

Execution

The programmer begins execution of a debug run by typing in the name of the LMN he has formed, followed by a period and a ⏎. This tells the system to fetch and begin execution of the specified LMN in the logon account.

As soon as the system starts execution of the LMN, the debugger takes control of program execution, and prompts the user with

SOURCE FILE=

This requests that the programmer enter the name of the file to which he assigned the M:SO DCB at compile time, followed by ⏎.

The debugger then prompts with

DEBUG FILE=

This requests the programmer to enter the name of the file to which he assigned the M:EO DCB at compile time, followed by ⏎. The debugger is then ready to accept debugging commands, and will prompt with the > character for each command.

APPENDIX A. DEBUGGER MESSAGES

The debugger issues three kinds of messages: Command error messages, informational messages, and data messages. This appendix describes these three classes of messages, giving examples where appropriate.

Command Error Messages

The debugger issues a command error message whenever it is unable to successfully fulfill a command request. This may occur because the command instructs the debugger to perform an illegal action – such as INSERT a line-number that already exists – or because the command references some nonexistant line-number, location, or identifier, or because of a syntactical error in the command specification.

In all cases the debugger indicates the position in the command line at which the error was detected, by typing a dollar sign at that position. It then issues a brief explanatory message. The messages and their meanings are given in Table A-1.

Informational Messages

These messages inform the programmer of changes of status during a debug run. An informational message is generated at each breakpoint, at each entry to a procedure-name or statement if PLIST or SLIST are specified, and any time an error or abort condition occurs during the run.

Table A-1. Command Error Messages

Message	Meaning
BAD DATA NAME	A nonexistent data-name was referenced.
INVALID QUALIFICATION	Either 'IN' is missing from a command, or a qualifying section does not exist.
BAD VALUE	An alphanumeric literal is larger than the identifier in the command.
BAD NUMBER	A source line-number either exists when a request to insert it is given, or doesn't exist when it is given in a SOURCE, DELETE, or REPLACE command.
INVALID DATA TYPE FOR NUMERIC MOVE	An attempt has been made to use a numeric literal with non-numeric data. Note: The debugger considers floating-point data (comp-1, comp-2) to be nonnumeric.
NO ROOM IN TABLE	The break table for the type of break being set (data, statement, or procedure-name) is full. An existing break must be removed.
TOO MANY COMMANDS... TABLE OVERFLOW	The table of commands associated with breaks is full. The current break is rejected. Another break must be deleted to make room.
BAD COMMAND	The debugger is unable to recognize the command.

Break Messages

A break message is output each time program execution reaches a breakpoint. A break message consists of two parts: First, the breakpoint identification which informs the programmer that a particular breakpoint has occurred; and second, any commands associated with the breakpoint are typed out before they are executed.

The breakpoint identification consists of an identification as to type of break, followed by the name of the last paragraph or section entered, and the statement-id of the current statement. For data breaks, the identifier consists of the words 'DATA BREAK', followed by the name of the identifier which caused the break. For statement and procedure breaks the identification consists of the words 'BREAK AT'.

Commands associated with breakpoints are typed out one-per-line.

LISTFILE Message

The message output after a LISTFILE command follows the form

```
                 ┌FILE       ┌file-id      ┐
                 │DEVICE     │device-id    │
ASSIGNED TO      │ANS        └serial-number┘
                 │LABEL
                 └**UNASSIGNED**

                 ┌                ┌INPUT  ┐
CURRENTLY        │OPEN            │OUTPUT │
                 └CLOSED          └I/O    ┘
```

Error/Abort Handling

Error and abort messages are issued to notify the user of some unusual condition which requires programmer intervention. In all cases, the error/abort message is issued to inform the programmer what has occurred, and then a breakpoint message is issued to notify him of the current program status.

If the program is halted or aborted by either a STOP RUN, a non-COBOL I/O error, or an illegal transfer of control, a message of the form

```
EXIT DUE TO
┌STOP VERB              ┐
│M:XXX OR M:ERR         │
└NON-COBOL I/O ERROR     ┘
```

is issued.

One of three messages will be issued if a trap occurs. If the trap is due to bad decimal data, the message will be

```
DECIMAL TRAP
```

If the trap occurs while processing a debugger command, the message will be

TRAP IN DEBUGGER

All other traps receive the message

UNIDENTIFIED TRAP

If the break key is depressed while the debugger is executing, the message

BRK IN DEBUGGER

will be issued. This will be the case if an I/O error occurs that is not handled by the COBOL run-time I/O routine.

PLIST and SLIST Messages

When PLIST or SLIST are on, the messages output consist of the procedure-names or statement-ids, respectively. No qualification is supplied for paragraph-names. A statement-id is output in the same form as it should be entered:

line-number[.copy line number][(verb number)]

A maximum of 20 procedure-name and 40 statement-ids will be displayed.

Data Messages/Displays

These messages are simply the outputs from PRINT or PRINTX commands. The format of the message depends on the class of the identifier as defined in the COBOL program. The debugger classifies all identifiers as either alpha-numeric, numeric-display, numeric-packed, numeric-binary-integer, or miscellaneous.

Alphanumeric data is presented to the programmer with no conversion, sixty characters per line.

Numeric-display data is presented with no conversion, except that the sign of the data will be shown as a trailing character rather than an overpunch.

Numeric-packed data is presented in hexadecimal format. This avoids unnecessary decimal data exceptions that might occur if the identifier were not initialized.

Numeric-binary-integer data is presented as a 9-digit integer, preceded by a sign.

Miscellaneous data is presented in hexadecimal format, sixty characters per line.

Table A-2 shows the debugger classification given to all types of identifiers.

Table A-2. Identifier Classes

Identifier Type in Program	Debugger Data Class
GROUP ITEM	Miscellaneous
ALPHANUMERIC (PIC X)	Alphanumeric
ALPHABETIC (PIC A)	Alphanumeric
NUMERIC EDITED	Alphanumeric

Table A-2. Identifier Classes (cont.)

Identifier Type in Program	Debugger Data Class
NUMERIC (PIC 9)	
DISPLAY	Numeric-display
PACKED (COMP-3)	Numeric-packed
INTEGER (COMP)	Numeric-binary-integer
INDEX	Numeric-binary-integer
FLOATING-POINT-SHORT (COMP-1)	Miscellaneous
FLOATING-POINT-LONG (COMP-2)	Miscellaneous

APPENDIX B. A SAMPLE DEBUGGING SESSION

This example shows some of the steps involved in actually debugging a small COBOL program. The sample program is a small edit-and-batch-balance program, assumed to exist in a file named EDIT-PROGRAM. Also assumed is that a file named EDIT-ASSIGNMENTS contains the !ASSIGN commands necessary to run the job, and that all the input files for the program exist.

To keep the explanations fairly straightforward, while still giving enough examples to provide a base starting point for a first-time debugger user, the following conventions are used:

1. Each line of terminal input and output is shown with a line number to its left. These numbers are not part of the input or output, but serve as reference numbers for the explanations.

2. Explanations are given following groups of input and output lines, rather than after each line.

```
1   XEROX CP-V AT YOUR SERVICE

2   ON AT 09:29 JUL 27, '73

3   LOGON PLEASE: TESTING, SEYMORECLEA

4   !SET M:EO DC/EDIT-DEBUG

5   !SET M:SO DC/EDIT-PROGRAM

6   !COBOL EDIT-PROGRAM OVER EDIT-ROM,LP

7      E00 COBOL

8       OPTIONS?

9   $COBOL TEST,GO

10    miscellaneous compiler output
```

Lines 1 through 3, up to the colon following LOGON are output by the CP-V monitor when the programmer is connected to the system. On line 3, the programmer enters his account, TESTING, and his name, SEYMORECLEA, to log on to the system. On lines 4 and 5, the programmer enters the SET commands needed so that the COBOL compiler will know where to write the debug file and the source output file. Note that to conserve file space, the programmer has chosen to make the source output file replace the source input file. Line 6 tells the monitor to call in the COBOL processor, with source input from the file EDIT-PROGRAM, with the object module being placed on the file EDIT-ROM, and with any listings generated by the compiler directed to the line printer. Lines 7 and 8 are output by the COBOL compiler. The COBOL compiler prompts for its control command with the dollar sign ($) on line 9, and the programmer then responds by entering a COBOL control command that specifies that this is a TEST mode compilation, that an object module is to be produced through the M:GO DCB, and that a data map is to be written through the M:LO DCB. The compiler output referenced on line 10 consists of a heading, a copy of the COBOL control command, any diagnostics issued, and a summary of number of diagnostics and severity level.

```
11  !LINK EDIT-ROM;BLIB:.CDBGLIB,BLIB:.COBLIB OVER EDIT-LMN

12     LINKING EDIT-ROM

13     LINKING BLIB:

14     LINKING BLIB:
```

On line 11 the programmer enters a LINK command, instructing the loader to form a load module from the object module named EDIT-ROM, satisfying external references (REFs) in the object module from the two libraries named BLIB: in the accounts CDBGLIB, and COBLIB, and to put the output load

module in the file named EDIT-LMN, replacing any file previously named EDIT-LMN. Lines 12 through 14 are information messages from the LINK processor. Link also will report the error severity level of each control section (CSECT) and dummy control section (DSECT) it processes. Since the number and position of these messages is unpredictable without a detailed knowledge of the program structure, they are not shown here.

```
15 !EDIT-LMN.

16  COBOL DEBUG HERE

17  SOURCE FILE=EDIT-PROGRAM

18  DEBUG FILE=EDIT-DEBUG
```

Line 15 tells the system to start execution of the load module named EDIT-LMN. On line 17 the debugger requests the name of the source file, and the programmer enters EDIT-PROGRAM. On line 18 the debugger asks for the name of the debug-file for this program, and the programmer replies with EDIT-DEBUG.

```
19 >SETFILES EDIT-ASSIGNMENTS

20 >AT WRITE-OUTPUT;PRINT EDITED-HOURS;STOP

21 >GO
```

The programmer tells the debugger to make the DCB assignments as specified in the file EDIT-ASSIGNMENTS. He then sets a breakpoint at the procedure name WRITE-OUTPUT, with a printout of the data item EDITED-HOURS, and a program halt to be executed when he reaches the breakpoint. He then instructs the debugger to begin executing his program.

```
22 DECIMAL TRAP

23 BREAK AT ADD-TO-TOTALS 683
```

A decimal trap occurs while the program is executing. The last procedure entered was ADD-TO-TOTALS, and the statement being executed when the trap occurred was on line 683.

```
24 >SOURCE 683

25              ADD 1 TO RECORD-COUNT

26 >PRINT RECORD-COUNT

27 D1E4D5D240

28 >SET RECORD-COUNT = 0

29 >GO
```

The programmer looks at line 683 of his program and decides that the trap must be due to bad data in RECORD-COUNT. He prints RECORD-COUNT, and discovers that the data is indeed bad. He then sets RECORD-COUNT to zero, and continues.

```
30 BREAK AT WRITE-OUTPUT 987

31 >PRINT EDITED-HOURS;

32 9999

33 >STOP
```

The program reaches the breakpoint set at WRITE-OUTPUT and executes the commands associated with the break. The programmer then looks at the printout of EDITED-HOURS, and decides that no error has been made.

```
34 >RUN

35 EXIT DUE TO STOP RUN

36 >END

37 !OFF
```

The programmer tells the debugger to remove all breakpoints and continue executing the program. The program runs to completion, the debugger notifies the programmer. The programmer terminates the debug run, and logs off the computer.

XEROX

Reader Comment Form

We would appreciate your comments and suggestions for improving this publication

Publication No.	Rev. Letter	Title	Current Date

How did you use this publication?

☐ Learning ☐ Installing ☐ Sales
☐ Reference ☐ Maintaining ☐ Operating

Is the material presented effectively?

☐ Fully Covered ☐ Well Illustrated ☐ Well organized ☐ Clear

What is your overall rating of this publication?

☐ Very Good ☐ Fair ☐ Very Poor
☐ Good ☐ Poor

What is your occupation?

Your other comments may be entered here. Please be specific and give page, column, and line number references where applicable. To report errors, please use the Xerox Software Improvement or Difficulty Report (1188) instead of this form.

Your name & Return Address

Thank You For Your Interest. (fold & fasten as shown on back, no postage needed if mailed in U.S.A.)

CUT ALONG LINE

PLEASE FOLD AND TAPE –
NOTE: U. S. Postal Service will not deliver stapled forms

FOLD ALONG LINE

FIRST CLASS
PERMIT NO. 39531
WALTHAM, MA
02154

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

FOLD ALONG LINE

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

22021, 2C1078, Printed in U.S.A.

XQ22, Rev. 0