HONEYWELL

CP-6 DELTA REFERENCE MANUAL

SOFTWARE

CP-6

DELTA REFERENCE MANUAL

SUBJECT

Description of the DELTA, Debugger for Programs Produced by Non-interpretive Language Processors

SOFTWARE SUPPORTED

DELTA C00 on Operating System C00.

ORDER NUMBER

CE39-03

March 1985

Honeywell

PREFACE

This manual is a reference document for DELTA, the Honeywell CP-6 universal debugger. DELTA is the debugger used for all programs produced by non-interpretive language processors (those that produce code which is LINKed and stored).

This reference describes in encyclopedic fashion the language elements, commands and features of DELTA. It is intended for use by:

o The experienced system programmer.

o The application programmer who is already familiar with DELTA. It is assumed that this reader has read the DELTA appendix contained in the appropriate programming language manual (e.g., the CP-6 COBOL or FORTRAN Programmer Guide or the CP-6 RPG Reference Manual).

The Los Angeles Development Center (L.A.D.C.) of Honeywell Information Systems Inc. has developed Computer Aided Publications (CAP). CAP is an advanced document processing system providing automatic table of contents, automatic indexing, format control, integrated text and graphics, and other features. This manual is a product of CP-6 CAP.

Readers of this document may report errors or suggest changes through a STAR on the CP-6 STARLOG system. Prompt response is made to any STAR against a CP-6 manual, and changes will be incorporated into subsequent releases and/or revisions of the manuals.

The information in this publication is believed to be accurate in all respects. Honeywell Information Systems cannot assume responsibility for any consequences resulting from unauthorized use thereof. The information contained herein is subject to change. New editions of this publication may be issued to incorporate such changes.

The information and specifications in this document are subject to change without notice. This document contains information about Honeywell products or services that may not be available outside the United States. Consult your Honeywell Marketing Representative.

©Honeywell Information Systems Inc., 1985 File No.: 1W13, 1W23 CE39-03

Table of Contents

Tables:

About this Manual

The contents of this manual are grouped into the following nine sections and two appendixes.

Section 1, Introduction

- Introduces DELTA, and the repertoire of DELTA commands.

Section 2, Prerequisite Information

- Defines DELTA-specific terms and describes the conventions used in the syntax of DELTA commands.

Section 3 Through 8

- Each section explains one of the six functional use categories and its commands.

Section 9

- Explains how to debug programs running in the FEP.

Appendix A, DELTA Distinguished Names

- Identifies and explains the hardware or system oriented entities that have been assigned names which may be in the syntactical components of several of DELTA's commands.

Appendix B, Assembler Instruction Formats for Patching

- Identifies and explains the exceptions to the standard GMAP6 instruction format for 1-word instructions.

Notation Conventions

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

Notation Conventions Table

Notation	Description
Lower-case Letters	Lower-case letters indicate that the element is a variable, to be replaced with the desired value.
CAPITAL LETTERS	Capital letters indicate a literal, to be entered as shown.
Brackets	An element inside brackets is optional. If elements are stacked vertically within brackets, the elements may be omitted or one of them may be entered. The brackets may be elongated to contain the stack of elements, or may be represented by vertically-stacked printed brackets. For example, the notation $\begin{bmatrix}\text{DISK}\\\text{TAPE}\end{bmatrix}$ means that the user may omit this entry, or may enter DISK, or may enter TAPE. When used to enclose keywords, brackets signify that the bracketed portion may be omitted, or truncated at any point. For example, the notation K[EY] means that the user may enter K, KE, or KEY.
Braces	Elements stacked inside a pair of braces identify a required choice. $\begin{Bmatrix}\text{id}\\\text{ALL}\end{Bmatrix}$ means that either a value for id or the word ALL must be entered. Alternatively, the vertical OR bar is used to separate the choices, thus: {id|ALL}

Notation Conventions Table (cont.)

Notation	Description
Horizontal Ellipsis	The horizontal ellipsis indicates that a previous bracketed element may be repeated, or that elements have been omitted. attachment[;attachment] ... means that one or more attachments may be entered, with a semicolon inserted between each attachment.
Numbers/Special Characters	Numbers that are not part of a parameter name, special symbols, and punctuation marks other than brackets, braces and underlines appear as shown in output messages and must be entered as shown when input. \f means that a format specifier (f) must be preceded by a \ character.

Section 1

Introduction

DELTA is the Universal Debugger for CP-6 programs. It is a special shared processor that, when invoked by a user, adds a system level dimension to the user's domain. This dimension includes levels of logic with a precise understanding of the programming, object, and machine languages. When invoked on behalf of a user, DELTA assumes both the user's identity and privilege levels.

DELTA is driven by a set of English-like statements called commands. These commands may be presented to DELTA directly through an interactive terminal or interspersed with IBEX commands in a batch job stream. In either case, DELTA translates the commands into the language of that execution level which is capable of carrying them out. Using DELTA, the user may:

- Conduct debugging sessions in the interactive or batch modes.
- Apply permanent patches to a run unit.
- Obtain current or post-mortem dumps of portions of a run unit.
- Examine the executing monitor.
- Examine any executing domain of the system.

External Debugging

DELTA is an external debugger. It does not increase the size of the executing run unit. Although it is associated with the run unit, it is completely independent of it. This is possible because:

1. A standard CP-6 language processor:
   - Outputs a compile unit containing processing instructions, data definitions and linkage declarations.
   - Outputs a highly encoded information map called a debug schema which defines the symbolic names given to all internal procedure labels, data names and external linkage names. The encoded schema defines the size, attributes and hierarchical shapes of all data names and any name scoping rules which apply either to block structured compile units or those which define variables in local, global or common sets. If the language processor permits nesting of subordinate procedure constructs (internal procedures), their hierarchical shapes are also encoded into the schema. The schema is stored along with the compile unit in a common file.
2. The LINK processor:
   - Links the individual compile units into the run unit, updates the debug schema from each compile unit to reflect the actual assignment of addresses, and writes all of the debug schema into the run unit file along with the program image.
3. The monitor:
   - Relays to DELTA any trap or fault occurring during execution of the run unit.

- o Relays to DELTA any exceptional condition which the run unit is not programmed to handle.

4. DELTA:

   - o Uses the debug schema to resolve symbolic references.
   - o Acts as an interface between the user, the run unit, and the monitor.

Languages Supported

DELTA may be used to debug programs produced by:

- o COBOL/COBOL-E
- o FORTRAN
- o PL-6
- o RPGII
- o GMAP6
- o FPL
- o Any language processor that produces code acceptable to the CP-6 LINK processor with or without debug schema information.

Program units produced by any of these language processors may be freely mixed, linked into a single run unit, and debugged under DELTA.

Capabilities Summary

DELTA may be executed in three major modes of operation. These are:

1. The Debugging Mode
2. The ANLZ Mode
3. The RUM (Run Unit Modification) Mode

Debugging Mode

DELTA allows debugging at the level of complexity that best suits the user's needs: at the source language level or at the machine language level.

Debugging at the source language level is accomplished using the terms and constructs of the language in which a particular program unit was written. Debugging at the machine language level is accomplished by using assembler mnemonics and special symbols. These two levels are not mutually exclusive; they may be used simultaneously. During a debugging session, program entities can be referred to using the terminology of either or both levels.

The following is a summary of the major capabilities provided by the debugging mode of DELTA. During a debugging session the user may:

- o Refer to the elements of the run unit as named in the source program. Using these names, the user may display and modify these program elements before, during and after program execution.
- o Determine the memory addresses of symbolic procedure and data names.

- Trace and record the flow of control of the executing run unit at several levels: on all entry points, on specified entry points, on all transfer points, or on specific locations. A history mode may be set such that trace information is saved for examination at a later time.

- Cause the flow of control to be conditionally or unconditionally altered based upon observation of the run unit in execution. Program execution may be interrupted at any time (using the break key) to allow interaction with DELTA.

- Search both procedure and data space for the occurrence of specific bit patterns within words of memory. A search may optionally be conducted under control of a mask and selective substitution may be performed under mask.

- Obtain snapshot or post-mortem dumps. Snapshot dumps may be taken at any time during a debugging session. Post-mortem dumps may be conditionally or unconditionally obtained by placing the proper command in the IBEX command stream for jobs executed in the batch mode in either the test or production environment. It is also possible for a run unit to associate DELTA programmatically within the exceptional condition handling routines, and to specify activities to be carried out by DELTA.

ANLZ Mode

The ANLZ mode of DELTA may be invoked only by those users who have been granted sufficient privilege by the System Manager. The ANLZ mode, when invoked, extends the visibility of DELTA to the internals of the CP-6 monitor itself. The user may observe the running monitor in execution, modify its data and procedure and, in general, perform any DELTA function which does not slow or halt system operation.

RUM Mode

By invoking the RUM mode of DELTA the user may:

- Permanently patch both the procedure and data of a run unit, primarily using the assembly instruction mnemonics.

- Modify certain of the control information in the run unit's HEAD record.

- Cause DELTA to provide a listing of all permanent patches.

Summary of Commands

Table 1-1 lists the entire repertoire of DELTA commands under the appropriate functional categories and sub-categories.

Table 1-1. DELTA Commands

Housekeeping		
	Input/Output Control:	
		COPY
		ECHO
		EOM
		OUTPUT
		PROMPT
		READ
		REPORT
		SYNTAX

Table 1-1. DELTA Commands (cont.)

Fault and TRAP Control:

- IGNORE
- KEEP
- ON ABORT
- ON EXIT
- TRAP

Addressing and Symbol Control:

- ALTERNATE VAR
- BYPASS
- DEFINE
- FORMAT
- RANGE
- SCHEMA
- USE

Stored Command Management:

- ACTIVE/INACTIVE
- DO
- KILL
- SAVE
- SHOW
- SILENT/UNSILENT
- UPDATE

Execution Control

Procedure Breakpoint:

- AT
- ON CALL(S)
- ON NODE(S)

Data Breakpoint:

- WHEN

Transfer of Control:

- ALIB
- BREAK
- EXIT
- GO
- GO TRAP
- UNSHARE
- XCON

Procedure Stepping:

- GO STEP
- GO TRAP STEP
- STEP

Table 1-1. DELTA Commands (cont.)
Execution Tracing HISTORY PLUGH TRACE
Memory Display and Modification Variable Oriented Commands: DISPLAY LET Word Oriented Commands: DUMP EVALUATE FIND MODIFY STORE Post Mortem Dump: PMD
Mode Control ANLZ RUM
Miscellaneous END HELP LIST PROTECT QUIT SAD UNFID XEQ

A complete list of DELTA's commands with acceptable minimal abbreviations can be found in Tables 1-2 through 1-7.

During a debugging session, the user may exercise almost total control over the execution of his run unit through the AT, ON, and WHEN commands. These three commands which are referred to as breakpoint commands are described in Section 4. The remainder of the commands in DELTA's repertoire are complements to these three commands.

The commands issued to DELTA are acted upon at different times. Some are executed immediately; others are stored for subsequent execution based upon some specific occurrence, and some are executed in conjunction with those which have been stored. DELTA commands can assume one or more of the following attributes:

1. Stored
2. Attached
3. Immediate
4. Toggled

Stored Commands

Stored commands are those which instruct DELTA to perform an action at some later time based upon the arrival of the Instruction Counter at a specific location or upon the occurrence of some specific event (the AT, ON, and WHEN commands primarily). When issued, these commands are assigned an identification number (id) either by the user or by DELTA (if the user does not), which is included in DELTA's report each time the command is activated. Stored commands may have other commands attached to them which are to be executed whenever the stored command is activated.
General Syntax

A stored command and its attachments are issued in the form:

```
    [id] [stored command][;attachment][;attachment] ... <CR>
```

The command line ends with a carriage return or end-of-record.

Commands can be continued onto a new line if the previous line ends with a semicolon and the new line begins with an attachment.

Example:

```
    10 AT READ-MASTER;DISPLAY NETPAY,GROSSPAY,DEDUCTIONS  <RET>
```

The example includes the following elements:

1. The stored command AT READ-MASTER was assigned an id of 10 by the user. If a stored command with the same id already exists, it will be replaced.
2. The single DISPLAY attachment will display three items (variables) within the user's run unit: NETPAY, GROSSPAY, and DEDUCTIONS.
3. The semicolon (;) is used to separate attachments.

In the above example, the programmer directs DELTA to set a breakpoint at the location "READ-MASTER" in the run unit. When the Instruction Counter reaches this location, DELTA will report the breakpoint at the user's terminal and then display the requested variables. If the programmer does not wish to interact with DELTA following the display, the command should be written:

```
    10 AT READ-MASTER;DISPLAY NETPAY,GROSSPAY,DEDUCTIONS;GO
```

With the command written in this form, DELTA will not stop after performing the display but will cause the run unit to resume execution as though the breakpoint had not occurred.

Conditional Execution

Format:

```
IF var logical_operator cons
```

Parameters:

var   is any position or location reference. Position and location references are described in Section 2.

logical_operator   is any of the following:

=	EQ	equal
>< <>	NE	not equal
<	LT	less than
>	GT	greater than
<= =<	LE	less than or equal
>= =>	GE	greater than or equal

cons   is any literal: octal, decimal, bit or character.

Description:

All stored commands may be formed to specify varying levels of conditional execution by using IF conditional. The IF conditional must be used as part of another command.

The value described by var is compared with the constant value. If the logical relation is true, then the stored command is reported and its attachments executed (if the condition was on a stored command), or the attachment is executed (if the condition was on an attachment).

Specification of an IF condition is allowed on almost all DELTA commands. Specific instances where the IF is not allowed are noted in the description of the commands.

Taking into account the IF condition, the complete specification of stored commands is:

```
[id] stored_command [IF var logical_operator cons]

  [;attachment [IF var logical_operator cons]

    [;attachment [IF var logical_operator cons]] ... ]
```

Rules:

1. The operators =, >=, <=, etc., cannot be entered from an online terminal unless "KILL EOM" has been specified.

Example:

```
20 AT ROUTINE10 IF NET_PAY EQ 0; DISPLAY DEDUCTIONS;
DUMP PAY_RECORD,20; GO
```

In this example, IF specifies conditional execution. When the ROUTINE10 breakpoint is reached, NETPAY will be examined to determine if its value is zero. If its value is not zero, DELTA will not report to the user that ROUTINE10 has been reached nor will any of the attachments be executed.

Attached Commands

Most DELTA commands may be issued as attachments to any stored commands. The exceptions are STEP, XEQ, and all stored commands.

Immediate Commands

A command (other than a stored command) assumes the immediate attribute whenever it is issued in-line. In the interactive mode this occurs whenever DELTA has issued its prompt character. In the batch mode this occurs whenever DELTA reads a command.

Example:

```
    >DISPLAY NET_PAY
```

In this context, DISPLAY is being used in an immediate way.

Toggle Commands

Certain of the commands in the housekeeping category set toggles within DELTA. These toggles may be reset by the KILL command. Those commands which set toggles are identified in Table 1-2 which is a functional summary of housekeeping commands. Tables 1-3 through 1-7 are functional summaries of the remaining five categories of DELTA commands. In these tables the brackets [] are used to indicate the minimum abbreviations allowed for each command.

Table 1-2. Housekeeping Commands

Command	Function
AC[TIVE]|IN[ACTIVE]	Activates or deactivates a single or a range of stored commands.
A[LTERNATE] V[ARIABLES]	Specifies alternate debug schema to be searched when an unqualified variable reference is not satisfied by searching the current schema.
BY[PASS]	Bypasses assembler program units during stepping. This command sets a toggle within DELTA.
C[OPY]	Causes DELTA output to be copied on the user terminal when the specified destination for output is other than the user terminal. This command sets a toggle within DELTA.

Table 1-2. Housekeeping Commands (cont.)

Command	Function
DE[FINE]	Associates a value or location with a symbol.
DO	Executes the attachments to a stored command or a group of commands identified by the SAVE command.
EC[HO]	Causes input to be echoed to an output device when DELTA input is from a device other than an on-line terminal. This command sets a toggle within DELTA.
EO[M]	Set or reset special activation (end of message) character set. This command sets a toggle within DELTA.
FO[RMAT]	Specifies default format for MODIFY and EVALUATE display output.
KE[EP]|TRAP|IG[NORE]	Direct DELTA's handling of asynchronous events and other exceptional conditions.
K[ILL]	Deactivates a toggle or removes a stored command or a range of stored commands.
O[N] A[BORT]	Specifies activities to occur upon abort.
O[N] E[XIT]	Specifies activities to occur upon normal exit.

Table 1-2. Housekeeping Commands (cont.)

Command	Function
OU[TPUT]	Specify destination for DELTA output.
PRO[MPT]	Sets the DELTA prompt character (default is >).
RA[NGE]	Specify range of offsets from defined symbol to be used for position reporting.
R[EAD]	Causes DELTA to read other than the normal input stream.
REP[ORT]	Directs DELTA's formatting of position reporting.
SA[VE]	Stores and remembers a single or a range of stored commands.
SC[HEMA]	Activates or deactivates schema usage or sets "current" schema. This command sets a toggle within DELTA.
SH[OW]	Displays the status of toggled options, keyword options or a single or range of stored commands and attachments.
SI[LENT]|UN[SILENT]	Activates or deactivates the reporting of a single or a range of stored commands.
SY[NTAX]	Allows specification of FORTRAN, COBOL, or PL-6 input syntax.

Table 1-2. Housekeeping Commands (cont.)

Command	Function
UP[DATE]	Updates stored commands or attachments of stored commands.
U[SE] N[ODE]	Activates schema(s) associated with a specific overlay node.

Table 1-3. Execution Control Commands

Command	Function
ALI[B]	Specifies return/altreturn from M$ALIB call to DELTA.
A[T]	Sets an instruction breakpoint.
B[REAK]	Passes control to user interrupt routine.
EX[IT]	Exits from a run unit invoked by M$LINK and returns to the linking program, or continues an M$LDTRC or M$SAVE.
G[O]	Proceeds with program execution.
G[O] S[TEP]	Goes to a specified location and executes one step.

Table 1-3. Execution Control Commands (cont.)

Command	Function
G[O] T[RAP]	Passes control to user's event handling routine when DELTA has been entered for an exceptional or asynchronous event.
G[O] T[RAP] ST[EP]	Same as GOTRAP except that one step is executed.
O[N] C[ALL]	Sets breakpoints on a specific procedure call.
O[N] [X] C[ALLS]	Sets breakpoints on all procedure calls. If X is specified, sets breakpoints only on external procedure calls.
O[N] N[ODE]	Sets a breakpoint on a specific overlay.
O[N] N[ODES]	Sets breakpoints on all overlays.
S[TEP]	Steps by statement or instruction.
UNSH[ARE]	Unshares an autoshared program and/or library so the user can have execution control.
W[HEN]	Sets a data breakpoint.
XC[ON]	Passes control to the user's exit control procedure simulating an exit condition.

Table 1-4. Execution Tracing Commands

Command	Function
H[ISTORY]	Displays contents of history buffer (filled by TRACE).
PL[UGH]	(Acronym for "Procedure List Used to Get Here"). Traces back through the automatic stack and lists the return addresses leading to the arrival at the current procedure point.
T[RACE] T[RANSFERS]	Traces all transfer instructions.
T[RACE] [X] C[ALLS]	Traces entry to all procedures. If X is specified, trace entry to external procedures only.

Table 1-5. Memory Display and Modification Commands

Command	Function
D[ISPLAY]	Displays the value of a variable or the contents of an address.
DU[MP]	Dumps a specified range of memory in octal or hexadecimal format. Optionally allows no suppression of duplicate lines. Optionally provides ASCII translation.
E[VALUATE]	Evaluates an expression and reports its value in a specified format. Reports the address of a program entity by segment and offset.

Table 1-5. Memory Display and Modification Commands (cont.)

Command	Function
F[IND]	Searches memory under mask and optionally substitutes under mask.
L[ET]	Changes the value of a variable or the contents of an address.
M[ODIFY]	Displays the contents of an address and optionally replaces it with new contents.
PMD	Dumps specified portions of a program which terminates abnormally.
STO[RE]	Modifies a range of memory. Optionally performs the modification under mask.

Table 1-6. Mode Control Commands

Command	Function
AN[LZ]	Associates the schemas for the CP-6 Monitor, or associates the schema from the specified file, and sets DELTA's domain of reference to that of the running monitor, a specified system dump file, or the running program.
RU[M]	Invokes the Run Unit Modification mode, optionally specifying the number of buffers to use for faster I/O (up to 10, default is 5).

Table 1-7. Miscellaneous Commands

Command	Function
EN[D] or Q[UIT]	Unconditionally exits to the command processor.
HELP	Provides HELP via the HELP facility.
LI[ST]	Lists changes made during Run Unit Modification.
PROT[ECT]	Sets Protect mode (disallows LET, MODIFY store).
SAD	Special Access Descriptor allows addressing through a Monitor descriptor for privileged users.
UNF[ID]	Performs M$UNFID on specified DCB.
X[EQ]	Executes a GMAP6 assembler instruction.

Invoking DELTA

DELTA can be invoked by entry of any of the following commands:

!S[TART] fid U[NDER] DELTA
    to start a program with DELTA associated.

!!DELTA
    to associate DELTA with a running program at any time, or after a program aborts.

!DELTA
    to use DELTA in calculator mode (i.e., no program associated).

!PMD
    to perform post-mortem dumps.

as explained in the following subsection.

DELTA may be entered at three times during the life of a program: as it starts execution, during execution and after a program aborts. DELTA may also be called in stand-alone mode, and at run-time by a user program via the M$ALIB monitor service. The methods of invocation and uses are discussed below.

Initiating A Program with DELTA Associated

Most debugging sessions are begun by starting the program to be debugged under DELTA. This is accomplished in two ways, both are commands in IBEX.

```
    !S[TART] fid U[NDER] [DELTA]
```

This command brings a run unit named fid into memory and prepares it for execution. DELTA is entered with the user program ready to run. DELTA prints the current value of the instruction counter, the program start address. For example:

```
    !START COBOLRU UNDER DELTA

    DELTA B03 HERE IC = PROG:0 [ENTRY]
```

This method can be used online and in batch. The user may now use any of the DELTA commands, or just say GO to begin execution. IC is the instruction counter and indicates the address that is about to be executed.

If the user's program reads the invocation line from B$JIT.CCBUF or depends on IBEX to implicitly set DCBs, the start command should not be used. Instead the IBEX command U may be used. The U implies the words UNDER DELTA on the next command line. For example:

```
    !U
    !MYCOMP.ME ON OUFILE,LP (LS,OU)
    DELTA B03 HERE IC = MAIN:0 [ENTRY]
```

In this case, the invocation line that is placed in B$JIT.CCBUF will be the same as if DELTA were not associated.

Post Association of DELTA

DELTA may be associated with a running program at any time. This is useful when a program appears to be looping or is in a bad or unexpected state, and the user wants to interrupt execution and see what the IC value is, or look at program variables. To associate DELTA after the program has started, the user types CTRL Y. IBEX prompts with a double bang (!!). The user types DELTA to associate DELTA with the interrupted program. When DELTA is associated, if the program is autoshared, the user should use the UNSHARE command if any modification of procedure (including breakpoints) is to be done. For example:

```
    !MYPROG.
```

The program does not prompt as expected; a loop is suspected. The user enters <CTRL><Y>.

```
    !!DELTA
    DELTA B03 HERE IC = INITVALS:54,,.3 [ASSIGNMENT]
```

This method is only available online.

When a user program aborts, IBEX holds the image of the run unit in memory. The user can associate DELTA with the image by typing DELTA immediately after the abort message. For example:

```
    !MYPROG.
          memory fault  (IBEX reports program abort due to
                                      memory fault)
    !DELTA
    DELTA B03 HERE IC = SUBPRO6:73,,.5 [INPUT/OUTPUT]
```

DELTA reports the IC value at the time of the fault. Any DELTA command can be issued, and the program can be continued by the GO command.

No Run Unit Associated

DELTA can be entered alone, i.e., with no program associated. This is sometimes referred to as calculator mode. DELTA is usually used to evaluate expressions when entered in stand-alone mode, although small programs in GMAP6 instruction format may be entered. For example:

```
    !DELTA
    >DELTA B03 HERE - NO RU ASSOCIATED
    >EV 250 +.13\U
    = 261
    >.101\C
    ='A   '
    >M .30 LDQ 43,DL
    >M .31 0
    >GO .30
       IPR fault @ .31/0
    >D $Q\U
     $Q=43
```

The procedure space available to the user when DELTA is in the stand-alone mode extends from location .20 to location .2000; locations .0 to .17 are reserved for use by DELTA. KILL EOM is the default in stand-alone mode. To enable it, the user can use the EOM command.

Programmed Association of DELTA

DELTA may also be associated by a running program via the M$ALIB monitor service call. Using this service a command may be passed to DELTA for execution. This is useful to an exceptional condition handling routine in a production program. If, for example, the program has entered the XCON routine due to bad data in an input buffer, a DUMP command may be passed to DELTA to print the offending buffer.

Communications with DELTA

DELTA uses the prompt character '>' when it is waiting for the user to enter a command at the terminal. All commands should be ended with the <RETURN> key on the keyboard.

All commands, except for the open form of the MODIFY command, will error if ended with <LINEFEED>. When <LINEFEED> is used with the open form of the MODIFY command, the next memory cell will be opened for modification.

By default, DELTA uses a special End-of-Message (EOM) character set. These are characters which terminate the mode and activate DELTA. The complete list of DELTA EOM characters includes T, <LINEFEED>, <TAB>, <RETURN>, [, {,], }, /, and =. Most of the special EOM characters are used only when debugging at the assembly or machine language level. However,] and }, which have the same effect, are a shorthand for the STEP command and are very convenient for stepping through a program. In order to enter any of these characters without activation to DELTA, for instance in a character string constant for the LET command, DELTA's special activation set must be turned off. This is done with KILL EOM command. The EOM command turns the special activation set back on. KILL EOM is the default for stand-alone mode.

If a symbol name is used that matches one of DELTA's special names, contains a period and is not a PL-6 structure name, contains a colon, or begins with an *, the symbol name must be specified by enclosing it in quotes followed by an "S". For example:

```
    '$JIT'S
    '.LABEL'S
```

If DELTA is already associated, depressing the break key will cause the running program to be interrupted. DELTA will report the current IC position and prompt for input.

Section 2

Prerequisite Information

Symbolic Addressing

A data name, statement label, statement number or any symbolic name assigned by using the DEFINE command during a debugging session may be used as a symbolic memory reference. When a symbolic memory reference is made, DELTA determines the actual memory addresses associated with the symbol. The rules governing the formation of symbolic names differ among the various programming languages. The symbolic name "X" could be an external procedure name in PL-6, a file name in COBOL or an element of an array in FORTRAN. For this reason, language processors place symbols which refer to data addresses (data names) and those which refer to procedure addresses (statement labels, statement numbers) in different tables within the schema. DELTA must therefore determine which of these tables is to be searched when resolving a symbolic reference into a memory address.

Resolving A Symbolic Reference

There are two classes of symbolic references:

1. A LOCATION REFERENCE refers to a memory address which contains data.

2. A POSITION REFERENCE refers to a memory address that contains an executable instruction.

DELTA does not require any special user effort to distinguish these references. In some cases, however, it is necessary to override DELTA's assumptions about a particular symbolic reference. This is explained in the commentary on those commands where it applies.

Symbolic Location Names

The following paragraphs will name and describe data entities as they are known to DELTA. These names will be used throughout the remainder of this manual.

Scalars

A scalar is a single item of data, an element.

Structures

A structure is a hierarchical set of names that refers to an aggregate of data items that may have different attributes. The various members of the structure may be further identified through the assignment of member names. There are two kinds of structures: major and subordinate.

A major structure exists at the highest level of the data hierarchy. It is known as a Level 1 structure to DELTA and is not viewed as being dependent or related to any other data construct.

A subordinate structure exists at some lower level of the data hierarchy (meaning some level number greater than 1) and is always contained within some structure with a level number less than its own.

Elementary Item

An elementary item exists at the lowest level of the data hierarchy.

Array

An array is a contiguous, named series of data constructs all of which have identical sizes and characteristics. Because each data construct has the same name, they must be referred to by their position within the array for unique reference. DELTA recognizes all forms of arrays exactly as they are declared by each of the CP-6 language processors. The rules that govern subscripting and/or indexing in the language through which the array was declared are recognized and honored by DELTA in any symbolic location reference.

Pointer

A pointer is a location reference construct used to specify a particular address in memory. There are several kinds of pointers. They will be discussed in later paragraphs in the context to which they apply.

Variable

A variable is the name used in any general reference to all of the data entities named above. Wherever a distinction in their treatment by DELTA is important, they will be referred to by their proper names.

Symbolic Position Names

The following paragraphs name and describe procedure entities whose names may be used in symbolic position references. They also identify those whose names may be used to qualify both position and location references.

Node

A node is an element in an overlay structure. A node name is the highest level qualifier that can be used in an address reference. Node applies only to overlaid programs.

External Compile Unit

An external compile unit (ECU) is the unit of compilation or assembly — the input to the LINK processor. An ECU name may be used to qualify both position and location references. Note that the ECU name is that name defined within the procedure code and not the name given to the object unit file in which it is stored.

Internal Program Unit

An internal program unit (IPU) is a logical block within an external compile unit which limits the scope of both procedure names and data names. An IPU name may be used to qualify both position and location references.

Entry Name

An entry name is an externally known procedure location within an external compile unit. There are primary entry names and secondary entry names. A primary entry name is one which names an external compile unit. Both may be used to qualify position and location references.

Statement Label

A statement label names a position within an ECU or IPU that may be used as the target of a transfer of program flow.

Statement Number

A statement number is that number assigned by a language processor to each statement that it translates into object code. DELTA recognizes statement numbers as valid, symbolic position references.

Substatement

A substatement is a division of a statement.

Offset

An offset is the octal or word offset from the specified statement/substatement. When a substatement is not specified, an offset must be preceded by two commas.

Format For Position

Position is used in the AT, GO, and GOSTEP commands and sometimes in the MODIFY command. The format for position can be one of the following:

o [ECU]:statement number,substatement,offset

  For the AT, GO, and GOSTEP commands, if ECU is omitted, the colon is optional. For the MODIFY command, the colon is required. The offset may be specified in octal or decimal. For example,

  :10,1,.1

o ECU{:INTERNAL ENTRY, ... }:label, substatement, offset.

  Label must be fully qualified by all INTERNAL ENTRIES necessary.

Example:

```
PROGNAME:PROCA:PROCB:LABEL1
```

- ECU + offset

```
PROGX+.54
```

- octal address

The octal address must be preceded by a period to be octal.

```
.2004
```

Note:

1. 2004 will be interpreted as:
   a. statement 2004 if used with an AT, GO, or GOSTEP command
   b. decimal location 2004 (octal location .3724) if used with any other command.

Symbolic Address Qualification

The following six levels of qualification may be required by DELTA to accurately resolve a memory reference:

1. Domain (XDELTA only)
2. Node name (overlaid program only)
3. External compile unit name (ECUNAME)
4. Internal program unit name (IPUNAME)
5. Major structure name (MS)
6. Subordinate structure name (SS)

Most qualification levels are summarized in Table 2-1.

Table 2-1. Summary of Symbolic Qualification

Qualifer	How specified	Example	Default to: *
NODE NAME	Use Command	USE NODE ABLE	Current Node
ECU	Name of ECU	ECUNAME:	Current ECU
IPU	Name of IPU	IPUNAME:	Current IPU
MS **	Name of MS	MSNAME.	No Default
SS	Name of SS	MSNAME.SSNAME	No Default

* For convenience DELTA recognizes what are termed the current node, current external compile unit, and current internal program unit. They are determined by the contents of the Instruction Counter at the time that the symbolic reference is resolved. Note that position references are implied by a colon (:).

Table 2-1. Summary of Symbolic Qualification (cont.)

** DELTA requires that all symbolic references to structures of a data hierarchy be qualified with the name of each structure to which it is subordinate (e.g., to qualify to the level of an elementary item within a three-level structure: MSNAME.SSNAME.VARIABLE).
 	
Note that multiple levels of structure qualification are both allowed and required to whatever level the elementary item is embedded:
 	
`MSNAME.SSNAME.SSNAME.SSNAME.VARIABLE`
 	
`LEVELS ->  1      2      3      4      5`

Pointer Qualified References

A pointer qualified reference has the form:

```
pointer -> variable
```

where pointer is a memory word which contains the address (in pointer format) of an entity in memory.

variable   is a based structure, array or scalar (based variable).

If the based variable was declared with an implicit pointer, DELTA is aware of the relationship between the pointer and the based variable and will automatically use the implied pointer in resolving memory references. The implied association may be overridden, however, by explicit pointer specification.

Assume that ARRAY1 is an array of one dimension. It is a based variable with an implied pointer called ARRAYPTR. Then:

Command	Explanation
DISPLAY ARRAY1(3)	Display the third element of the array using ARRAYPTR to determine the address of the array (implicit qualification).
DISPLAY ARRAYPTR->ARRAY1(3)	Same. Qualification was unnecessary.
DISPLAY BASEPTR->ARRAY1(3)	Display the third element of the array. Use "BASEPTR" to determine the address of the array (explicit qualification).

Multiple pointers may be used in a pointer qualified reference.

Example:

```
DISPLAY BASEPTR1->BASEPTR2->ARRAYPTR->ARRAY (3)
```

In this example:

1. BASEPTR1 contains the address of BASEPTR2.
2. BASEPTR2 contains the address of ARRAYPTR.
3. ARRAYPTR contains the address of ARRAY.

Notes:

1. Every pointer in a pointer qualified reference must be in pointer format.

Descriptor Qualified References

All descriptors which are visible to a run unit may also be used to qualify memory references. Each of these descriptors is known to DELTA by a distinguished name and may be used as a pointer in much the same manner as a pointer variable or constant. All of the distinguished names known to DELTA are detailed in Appendix A. The more common ones are listed below.

Distinguished Descriptor Name	Refers To:
$LS0 or $ISR	Instruction Segment
$LS1 or $JIT	Job Information Table (JIT)
$LS3	Read Only Segment
$LS4 or $DS1	Automatic Segment
$LS5 or $DS2	Common Segment
$LS6 or $DS3	Dynamic Data Segment 3
$LS7 or $DS4	Dynamic Data Segment 4
$LS8 or $DS5	Dynamic Data Segment 5
$LS9 or $DS6	Dynamic Data Segment 6
$LS10 or $DS7	Dynamic Data Segment 7
$LS11 or $DS8	Dynamic Data Segment 8
$LS12	Null Segment

Whenever descriptor qualification is used, the distinguished name of the descriptor must be the leftmost qualifier in the memory reference.

Example:

Command	Meaning
DISPLAY $LS5->ARRAY1(3)	Display the third element of ARRAY1. The base address of the common segment is also the base address of ARRAY1.
DISPLAY $LS5->4->ARRAY1(3)	Display the third element of ARRAY1. Use word 4 in the common segment to determine the address of ARRAY1.

DISPLAY $LS2->0->NEXT->MS.SS.ITEM	Use word zero of the segment described by Linkage Segment Descriptor #2 to determine the address of the based pointer called NEXT. Use NEXT to determine the address of the Major Structure called MS. Display the elementary item called ITEM contained within the Subordinate Structure SS.

Section 3
Housekeeping Commands

The commands discussed under this heading are those which influence the behavior of the DELTA processor itself. Their purpose is to provide the greatest possible flexibility in specifying the manner in which to communicate with DELTA, how DELTA is to communicate with the user, how DELTA is to interact with a run unit and how DELTA is to deal with both predictable and unpredictable events which occur during the execution of a run unit.

All of the commands in this category affect the way in which DELTA behaves. Some set toggle switches which DELTA examines to determine whether or not a given activity is enabled or disabled. Others override certain default assumptions which are automatically established when DELTA is invoked. The default toggle settings and the default assumptions are those which are normally specified by the user. It is therefore quite probable that a debugging session can be conducted without using any of the commands in this section.

Housekeeping commands are divided into the following categories:

- o Input/Output control commands
- o Addressing and symbol control commands
- o Stored command management commands
- o Fault and trap control commands
- o Miscellaneous housekeeping commands

Input/Output Control Commands

The Input/Output control commands include the following:

ECHO, PROMPT, READ, OUTPUT, COPY, EOM, SYNTAX

Input/output control commands allow the user to specify alternatives to the default assumptions for the following questions:

1. From where are DELTA's commands to be READ?
2. Where are DELTA's diagnostics, messages and displays to be written?
3. Should commands read from other than the primary input stream be "echoed" on the output stream?
4. In an interactive session, if the output stream has been directed away from the interactive terminal should DELTA's output also be written to the terminal?
5. What prompt character do you wish DELTA to use?
6. How are symbols, expressions, and structure references specified to DELTA?

ECHO Command

Format:

EC[HO]

Description:

The user may specify that DELTA is to read its commands from some source other than the default stream (see the READ command). The ECHO command allows the user to see these commands at the interactive terminal. When ECHO is in effect, all input read by DELTA from other than the interactive terminal is written to the M$DO DCB.

Usage Notes:

1. ECHO is a toggle command. It is reset by the command:

   KILL ECHO

2. ECHO is the initial default.

PROMPT Command

Format:

PRO[MPT] char_string

Parameters:

char_string   is the desired prompt character string, from 1 to 28 characters. If all numeric characters or any non-alphanumeric are used, the entire string must be enclosed in quotes.

Description:

This command allows the user to change DELTA's prompt character (>) to another character or string of characters.

READ Command

Format:

R[EAD] [fid[,rec]]

Parameters:

fid   may be any valid CP-6 file identifier (fid). See the discussion on fids in the Programmer Reference Manual (CE40). Specifying the fid ME resets the input to the default command stream. When fid is a CP-6 managed file, the specification of a starting record number is allowed.

rec   can be either a record sequence number or, if the file is an EDIT compatible, keyed file, an EDIT line number. Specification of an EDIT line number must always contain a decimal point with the exception of record number zero (0) which is assumed to be edit line number 0.000.

Description:

Unless directed otherwise, DELTA reads from the normal program command stream. This is the interactive terminal during on-line sessions, the input job command stream when in batch, or the file of command input specified by the IBEX XEQ command in either situation. The DCB used by DELTA for reading its input is M$DELTA, one of the reserved system DCBs. A user program cannot affect this DCB; however, a user can direct DELTA's input stream prior to entry to DELTA through the IBEX SET command. Once DELTA is in control, DELTA's input stream may be re-directed by use of the READ command.

Example:

Directive	Explanation
READ MYFILENAME	Read from the file MYFILENAME starting with the first record.
READ DP#JRC/JEFFILE,5	Read from the file JEFFILE contained on #JRC beginning with the fifth record.
READ EFILE,6.	Read from the EDIT compatible file EFILE beginning with line number 6.000.
READ ME	Read from the default command input stream.

Usage Notes:

1. Upon reaching end-of-file or upon the occurrence of any unusual condition while reading, DELTA resets its input to the default command stream.

2. Operationally, the READ command acts as a SET command on DELTA's input DCB. It does not cause the <u>initiation</u> of any read operations; it simply alters the source of input for the next time that DELTA reads a command.

3. When an EDIT key is specified but is found not to exist, DELTA begins reading at the following record.

4. If fid is not specified, DELTA defaults to READ UC.

OUTPUT Command

Format:

OU[TPUT] [[ON|TO|OVER|INTO] fid]

Parameters:

[ON|TO|OVER|INTO] determines the disposition of an existing file of the same name as the fid.

fid is a valid file identifier.

Description:

DELTA'S output is always written through the M$DO DCB. The default destination differs depending upon whether the session is conducted on-line or off-line. The default destination for an on-line session is the user's terminal; for a batch session it is the line printer at the user's workstation of origin. The OUTPUT command overrides these defaults.

Example:

Command	Explanation
OUTPUT ON MYFILENAME	Write DELTA output on a new file called MYFILENAME.
OUTPUT OVER MYFILENAME	Overwrite the old file called MYFILENAME with DELTA output.
OUTPUT INTO MYFILENAME	Extend the old file called MYFILENAME with DELTA output.

Usage Notes:

1. The ON or TO option will cause an error if the fid references a file which already exists. The OVER option will cause the file to be overwritten if the fid references a file which already exists. The INTO option will cause the file to be extended if the fid references a file which already exists. In all cases, if the fid references a file and the file does not exist, it will be created. If none of ON/TO/OVER/INTO are specified, the default is ON.

2. DELTA does not exercise complete control over the M$DO DCB. The connection of the DCB may be changed by:

   o Using the IBEX SET command.

   o Using the M$OPEN or M$CLOSE monitor service in the program being debugged.

3. If the run unit being debugged also routes data through the M$DO DCB, DELTA's output will be intermingled with it. Should the run unit CLOSE the DCB, it will be implicitly reopened by the next write from DELTA using whatever parameters that remain.

4. If the run unit being debugged explicity OPENs the M$DO DCB, it must be able to handle the ALTRET condition "DCB is already open", since DELTA will have opened the M$DO DCB to write its greeting message when DELTA is first associated.

COPY Command

Format:

C[OPY]

Description:

DELTA's output is normally routed to the interactive terminal during an on-line session. The interactive user who has routed DELTA output to some other destination (e.g., a fid so that a copy of the session is maintained) may also wish to use the COPY command to see DELTA's output at the interactive terminal. By using COPY in conjunction with OUTPUT, the user directs DELTA's output to two places.

Example:

These examples of input/output control commands assume an on-line session.

Command	Explanation
OUTPUT ON LP	Direct DELTA's output to the line printer.
ECHO	Write any input not received from the default command stream to the M$DO DCB (in this example, to the line printer).

READ MYFILE	Read the next commands from the file called MYFILE. (Commands will be echoed on the line printer).
COPY	Direct DELTA's output to two places.
READ MYOTHERFILE	Read the next commands from the file called MYOTHERFILE. (These commands will now be written both to the line printer and the interactive terminal).

Usage Notes:

1. COPY is a toggle command. It is reset by the command:

   KILL COPY

2. KILL COPY is the initial default.

EOM Command

Format:

EO[M]

Description:

The EOM (end of message) command activates a special character set which signals end of message when using the MODIFY command and during program stepping. The EOM character set is shown in Table 6-1 which also shows sub-commands which may be used in place of the EOM characters.

Cross Reference:

See also the STEP and EVALUATE commands.

Usage Notes:

The EOM character set may be disabled with the KILL EOM command. The default is EOM unless you are in stand-alone mode.

SYNTAX Command

Format:

SY[NTAX] [compiler]

Parameters:

compiler   is one of the following:

```
F[ORTRAN]
FP[L]
C[OBOL]
P[L6]
R[PG]
G[MAP]
PL1
```

Description:

The SYNTAX command tells DELTA how symbols, expressions, and structure references are specified.

The initial default of SYNTAX is the compiler that produced the main program of the run unit being debugged. If compiler is omitted, the initial default compiler will be used. As far as DELTA is concerned PL6, RPG, GMAP6 and PL1 are identical and COBOL and FPL are identical; thus, three types of distinct SYNTAX can actually be specified: SYNTAX PL6, SYNTAX COBOL, and SYNTAX FORTRAN.

The character set for symbols in each SYNTAX specification is identical to the character set allowed in the language definition. The major difference between SYNTAX COBOL and SYNTAX PL6 is that the embedded dash or minus sign (-) is allowed in SYNTAX COBOL. This difference gives rise to the difference in expressions in SYNTAX COBOL and SYNTAX PL6. In SYNTAX PL6, the expression operators (-,+,*,%,>>,<<) must separate the expression elements (symbols, constants, etc.) with no separating blanks. Conversely, in SYNTAX COBOL the expression operations must be separated from the expression elements by leading and trailing blanks.

In SYNTAX PL6, structure items are separated by periods, in order of major level identifier followed by minor level identifiers. A SYNTAX PL6 structure reference must be fully qualified.

In SYNTAX COBOL, structure items are separated by the keywords IN or OF. The major level structure item is specified last, as is natural in the COBOL language. A COBOL structure reference may take one of three forms. In the first type of reference, only one identifier is specified. The reference will be satisfied by the first variable name that matches the given identifier, regardless of actual structure level. In the second type of reference, two identifiers are specified. The major structure name must be the second identifier specified. In the third type of reference, any number of identifiers may be specified. The reference must be fully qualified by specifying all low level references and the major structure name must be the last identifier specified.

The SYNTAX command also tells DELTA how to interpret decimal constants for the LET command. For SYNTAX PL6, .octal-digit-string is considered to be a right justified octal constant. For SYNTAX FORTRAN, .decimal-digit-string is considered to be a floating point (real) constant. For SYNTAX COBOL, .decimal-digit-string is considered to be a fixed point decimal constant. (See the LET command for more information about constants.)

Example:

```
    SYNTAX   COBOL
    DISPLAY  D IN A
    SYNTAX   PL6
    DISPLAY  A.B.C.D
    SYNTAX   FORTRAN
    LET REAL 3E21
```

Usage Notes:

1. When SYNTAX COBOL is in effect, it is possible to specify a non-fully qualified reference that DELTA considers valid, but that the COBOL compiler will consider ambiguous. The ambiguous reference which DELTA finds is undefined.

2. The SYNTAX command is likely to be useful only when a run unit consists of object units produced by different compilers.

3. The SYNTAX option of the SHOW command displays the current setting of the SYNTAX mode.

Addressing and Symbol Control Commands

The addressing and symbol control commands include the following:

SCHEMA, USE NODE, ALTERNATE VARIABLES, FORMAT, DEFINE, RANGE, REPORT.

The commands in this category allow the user to address every area of memory in a user domain accessible to a run unit. Certain of the commands exist as debugging conveniences designed to make the task of communicating with DELTA as easy as possible. The DEFINE command for example, assigns symbolic names to areas of memory which were not symbolically defined during the compilation or assembly process. It relieves the user of the responsibility of having to remember memory addresses which may become important during a debugging session.

The SCHEMA and ALTERNATE VARIABLE commands are used to influence DELTA's default assumptions concerning unqualified, symbolic references within run units that contain more than one external compile unit (ECU).

The USE command has two purposes:

1. To specify to DELTA which node of an overlay program is to be assumed as the starting point for future symbolic references.

2. To specify to XDELTA the domain of reference to be used.

The REPORT command specifies the way that position references are reported by DELTA.

SCHEMA Command

Format:

SC[HEMA] [position|fid]

Parameters:

position may be any position reference. If position is omitted, general schema usage is reactivated with the defaults in effect as described above.

fid is a valid file identifier.

Description:

This command is used:

1. To reactivate schema usage that has been deactivated by the SCHEMA option of the KILL command.

2. To specify the current schema for all nonqualified position or location references.

There is always some default in effect regarding the current schema unless general schema usage has been deactivated by the SCHEMA option of the KILL command. These are:

1. In the debug mode, the current schema is the one associated with the ECU indicated by the contents of the Instruction Counter (IC).

2. In the RUM mode, the current schema is the one associated with the ECU which contains the start address of the run unit.

Example:

Directive	Explanation
SCHEMA	Reactivate general schema usage and default to the current schema as per IC.
SCHEMA PROBNAME:556	Consider the schema associated with PROBNAME:556 the current schema.

Cross Reference:

See the ALTERNATE VARIABLES command.

USE NODE Command

Format:

```
U[SE]   {N[ODE] nodename}
        {[CURRENT]      }
```

Parameters:

nodename   specifies the name of an overlay node. When CURRENT is specified the schemata for the overlay nodes currently in memory are re-established.

Description:

The USE command is used to specify the name of a program overlay when running in the debug or RUM mode of DELTA.

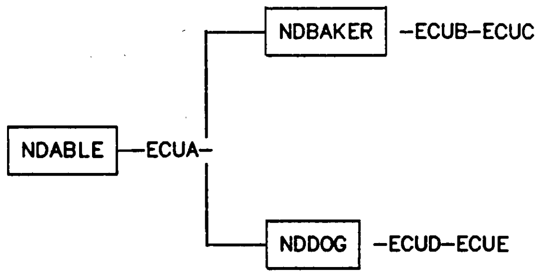
When the command is issued, the schemata associated with the overlay node and its backward path are activated so that symbolic references may be made using symbols defined within the overlay schemata.

Special Considerations:

The USE NODE command sets breakpoints at procedure positions within overlay nodes that are not in memory. This command prevents waiting for those operations until the overlay node is loaded into memory. DELTA does not, however, prevent an attempt to modify a location by using a symbolic location reference in either the LET or MODIFY command. If the overlay node named in the USE NODE command is not actually in memory, the contents of memory which that symbol would occupy if it were in memory will be changed and no warning would be issued by DELTA. To modify a location in a overlay node not in memory, the ON NODE command is used to cause a breakpoint when the overlay node is loaded. See the Usage Notes for more information.

Example:

Assume that the target run unit consists of five ECU's in the following tree structure:



where

NDABLE	is a node containing an external compile unit ECUA.
NDBAKER	is a node containing ECUB and ECUC.
NDDOG	is a node containing ECUD and ECUE.

Upon initial entry to DELTA, the user wishes to establish breakpoints at statement number 50 in ECUB, statement number 100 in ECUE, and statement number 50 in ECUA.

Command	Explanation
AT 50	Having entered DELTA, the root node (NDABLE) is in memory and ECUA is implicit; therefore, no qualification was necessary.
USE NODE NDBAKER	NDBAKER is the node name for the set of ECU's ECUB and ECUC.
AT ECUB:50	Qualification is required to distinguish statement number 50 in ECUB from statement number 50 in ECUC and ECUA.
USE NODE NDDOG	This specifies the node containing ECUE.
AT ECUE:100	ECUE is used to qualify the desired statement number.

Usage Notes:

1. When USE NODE is used in the RUM mode, the overlay is made to look as if it is in memory, i. e., instructions and data are brought in as well as schema. Any modifications made while USE NODE is in effect will be made to the location in the named overlay node, and not to the root node.

2. The USE NODE command is not allowed for FEP programs.

ALTERNATE VARIABLES Command

Format:

A[LTERNATE] V[ARIABLES] position

Parameters:

position   may be expressed as either a primary or secondary entry name.

Description:

This command is used to specify an alternate schema to be searched by DELTA whenever a search for a location (variable) reference is not satisfied within the current schema.

The command can be used to great benefit during debugging sessions whose target run units are composed of two or more ECU's. Through use of this command, references to both global and based variables may be made without ECU qualification.

Example:

Command	Explanation
ALTERNATE VARIABLES ENTRYNAME	Search the schema associated with ENTRYNAME whenever a search within the current schema has failed.

Usage Notes:

1. The ALTERNATE VARIABLES schema is always searched as a last resort when DELTA tries to satisfy a variable reference (MODIFY, LET, DISPLAY, FIND, STORE, DUMP commands).

2. For COBOL and FORTRAN programs, the initial default for position is the ECU that contains the run unit's start address. For all other programs, the initial default position is the entry name B_DELTA_D.

Cross Reference:

See the SCHEMA command (position option).

FORMAT Command

Format:

```
FOR[MAT] {M[ODIFY]  } \f
         {E[VALUATE]}
```

Parameters:

f   is one of the specifiers in Table 3-1.

Table 3-1. Format Specifiers

Specifier	Meaning	Example
A[R]	Display left 24 bits of a word as word-char-bit.	.35-2-5
B[IT]	Display in binary format.	'010110100'B
C[HAR]	Character.	'ABCD'
D[ESCR]	Descriptor.	.46000,BD=.75777-3, FL=.643,WSR=7,TY=0
EB[CDIC]	EBCDIC Character	'694E'
E[PTR]	Displays left half of word as ENTDEF+.offset[:stmnt#]	PROGB+.374 :27
F[LOAT]	Floating point binary. Single precision for 36-bit items, double precision for 72-bit items.	5.789604E+76
I[NSTR]	Assembly language instruction.	LDQ .1,DL

Table 3-1. Format Specifiers (cont.)

J[DE]	JIT Dot ERR. Displays error message for the value stored in JIT.ERR.	FMN-M00113-0 File does not exist
O[CTAL]	Octal digits with leading zeroes suppressed.	.1024
P[TR]	Pointer. word-char-bit, segid.	.35-2-7,$LS0
R[EL]	Relative. Primary ENTDEF+offset[,:stmnt#, substmnt, offset] or SYMDEF + offset.	PROGA+.6 :12,,.1(LOOP)
REM[EMBER]	Remember.	TEST:6(LABEL)[ASSIGNMENT](+.4)
S[BIN]	Signed binary (decimal).	-357
T[IME]	Convert UTS to display format.	13:52:36.82 06/25/79
U[BIN]	Unsigned binary (decimal).	357
V[ECTOR]	Vector.	.6245-0-0,$LS0,BD=.14-2, FL=.777,TY=NORMAL SHRINK
X	Hexadecimal.	'F100CS40D'X
X1	Pseudo-hexadecimal. Leading bit of each byte ignored.	'F0F8F6F4'X
Z[ERO]	Displays a word value in Octal with leading zeroes.	.000000001024

Description:

The FORMAT command establishes the default display for the EVALUATE and MODIFY commands. The normal display format for these commands is octal with leading zeros suppressed.

DEFINE Command

Format:

DE[FINE] symbol expression

Parameters:

symbol	is a symbol name up to 30 characters in length.
expression	is any address expression which results in a value. Overflow is ignored, and only the least significant 36 bits of the value are saved.

Description:

The DEFINE command associates a symbolic name with a value, allowing the symbol to be used whenever the associated value is desired. This is particularly useful for defining addresses in programs that do not have debug schema.

Example:

```
     DEFINE HEP .14136
     DEFINE HEPMORE HEP+.201
```

Usage Notes:

1. A DEFINED symbol may be removed with the DEF option of the KILL command.

2. DEFINE cannot be used to define replacement names for ENTDEFs. An ENTDEF (primary entry definition, secondary entry definition, ECU name) is a special entity which has schema associated. A symbol name defined with DEFINE has no associated schema. Therefore, while:

```
    MODIFY VERYLONGENTRYNAME:23
              and
    DEF VLEN VERYLONGENTRYNAME
```

   are meaningful, MODIFY VLEN:23 is not.

RANGE Command

Format:

RA[NGE] value

Parameters:

value	is any octal or decimal literal. The default is .7777.

Description:

Sets the maximum position range DELTA will use when determining whether to print a location as an octal location, or as the closest of a user DEF + offset or SYMDEF + offset.

Example:

```
    DEF A .30
    RANGE .40
```

When modifying location .70, DELTA will write A+.40, but modifying location .71 results in .71.

REPORT Command

Format:

REP[ORT] [compiler] [info]

Parameters:

compiler   is one of the following:

```
G[MAP]
P[L6]
F[ORTRAN]
FP[L]
C[OBOL]
PL1
R[PG]
```

If compiler is omitted, the info field applies to all compilers.

info   is one of the following:

```
A[LL]
D[EFAULT]
```

or any combination of O[FFSET, L[INE], or T[YPE], separated by commas. If info is omitted, the info field is treated as DEFAULT.

A complete position report consists of the following fields:

```
Primary  +.offset :line# (label),substmnt#,.offset [TYPE] /GMAP6 instruction
ENTDEF  \________/\___________________________________/\______/\______________/

Field        1                  2                         3           4
```

Display of fields 1 and 4 is enabled by specifying OFFSET.

Display of field 2 is enabled by specifying LINE.

Display of field 3 is enabled by specifying TYPE.

Field 4 is available only when the report is generated as the result of a STEP command.

Description:

The REPORT command controls the manner in which DELTA displays position references. DELTA normally displays position reference information based upon what compiler produced the object unit from which the current position is being displayed.

Example:

```
REPORT ALL
REPORT PL6 OFFSET, LINE
REPORT COBOL ALL
```

Usage Notes:

1. Schema is required for the REPORT command to have any effect. If schema has been KILLed, or if the object unit that contains the position reference being displayed by DELTA does not have any debug schema associated with it, the REPORT will be identical to REPORT GMAP, i.e.,

   def +.offset (label)

   where def is the closest primary ENTDEF; or if there are no ENTDEFs less than or equal to the position, def is a user defined symbol (subject to RANGE).

   label   is an exact matching secondary ENTDEF or SYMDEF.

2. The initial defaults for REPORT as a function of compiler type may be displayed by using the REPORT option of the SHOW command when DELTA is first entered.

Stored Command Management Commands

The stored command management commands include the following:

KILL/SHOW, SILENT/UNSILENT, ACTIVE/INACTIVE, UPDATE, SAVE, DO

This category of housekeeping commands allows the user to exercise control over and remain cognizant of up to 99 stored commands and to set and reset the status of those toggle options detailed throughout this section.

Within this category are commands which control the "noise level" at a terminal (SILENT/UNSILENT). Breakpoints which are serving a useful purpose but whose reporting is no longer necessary may be silenced.

Breakpoints which have been entered but are not useful in a given debugging situation may be temporarily inactivated and then activated again at a later time without having to re-enter them (ACTIVE, INACTIVE).

Commands which are to be executed often may be saved as though they were attached to breakpoint commands and directly invoked whenever necessary (SAVE).

The KILL command revokes actions which have been taken previously and the SHOW command displays both the status of toggles and the text of stored commands.

KILL/SHOW Command

Format:

```
{K[ILL]}  {id[-id]    }
{SH[OW]}  {id [TO id] }
          {keyword    }
```

Parameters:

id   is the identification of a stored command.

keyword   is one of the keywords in Table 3-2.

Table 3-2. Keywords Used with KILL and SHOW

Keyword	Meaning with Kill	Meaning with Show
AL[L]	Remove all stored commands.	Display status of all stored commands, toggle options, and modes.
A[LTERNATE] V[ARIABLES]	Discontinue use of alternate variables.	Show status of toggle and schema name if any.
AN[LZ]	Return to debug mode.	Show status of toggle.
A[TS]	Remove all AT breakpoints and their attachments.	Display all AT breakpoints and their attachments.
B[YPASS]	Do not bypass step reporting in assembler modules.	Show status of toggle.
C[OPY]	Discontinue COPYing.	Show status of toggle.

Table 3-2. Keywords Used with KILL and SHOW (cont.)

Keyword	Meaning with Kill	Meaning with Show
D[EF]	Remove a specific named DEFINED symbol.	Not applicable.
DEFS	Remove all DEFINED symbols.	Display all defined symbols.
DEL[TA]	Causes DELTA to be disassociated from the current run unit being debugged.	Not applicable.
EC[HO]	Discontinue ECHOing.	Show status of toggle.
E[OM]	Deactivate the EOM character set.	Show status of toggle.
F[ORMAT]	Set display formats for MODIFY and EVAL back to initial defaults.	Display current default formats for MODIFY and EVAL.
I[GNORE]	Not applicable.	Display which exceptional condition groups and/or names are being ignored.
K[EEP]	Not applicable.	Display which exceptional conditions will be intercepted and reported by DELTA.
O[N] A[BORT]	Remove ON ABORT breakpoint and its attachments.	Display ON ABORT breakpoint and its attachments.
O[N] C[ALLS]	Remove all ON CALL(S) commands and their attachments.	Display all ON CALL(S) commands and their attachments.
O[N] E[XIT]	Remove ON EXIT breakpoint and its attachments.	Display ON EXIT breakpoint and its attachments.
O[N] N[ODES]	Remove all ON NODE(S) commands and their attachments.	Display all ON NODE(S) commands and their attachments.
P[ROTECT]	Discontinue PROTECT mode.	Display PROTECT mode.
R[ANGE]	Not applicable.	Display value of range specification for relation position reporting.
RE[PORT]	Not applicable.	Display reporting mode for position reporting.
RU[M]	Return to debug mode.	Show status of toggle.
SAD	Not applicable.	Display special access descriptor number.
SA[VES]	Remove all SAVE commands and their attachments.	Display all SAVE commands and their attachments.

Table 3-2. Keywords Used with KILL and SHOW (cont.)

Keyword	Meaning with Kill	Meaning with Show
SC[HEMA]	Discontinue schema usage.	Display the position which defines the current schema, as set by the instruction counter or the SCHEMA command.
S[TEP]	Default to step by statement.	Show status of STEP mode.
SY[NTAX]	Not applicable.	Display which input syntax DELTA is currently accepting.
T[RACE]	Discontinue tracing.	Display all TRACE commands.
TRAP	Not applicable.	Display which exceptional conditions will be passed to the trap handler in the target run unit.
W[HENS]	Remove all WHEN breakpoints and their attachments.	Display all WHEN breakpoints and their attachments.
FEP Keywords:		
DEL[TA] fprg-res	Causes DELTA to be disassociated from the specified FEP program.	Not applicable
FP[RGS]	Not Applicable.	Displays the state of all existing FEP programs being debugged.
fprg-res	Not Applicable	Display the state of the specified FEP program.

Description:

The KILL command is used either to change the status of a toggle or to cancel a previously entered command.

The SHOW command is used to display the status of a toggle, classes of commands and attachments, and the current disposition of exceptional conditions.

SILENT/UNSILENT Command

Format:

```
{SI[LENT]  }  id [-id  ]
{UN[SILENT]}     [TO id ]
```

Description:

These commands control the reporting of active breakpoints. They silence redundant reporting once the user determines that a breakpoint or a range of breakpoints is achieving the desired results.

Example:

Command	Explanation
SILENT 10-15	Silence the reporting of the breakpoints in the range specified.
UNSILENT 10-15	Resume reporting of the breakpoints in the specified range.
SILENT 7	Silence stored command number 7.

Usage Notes:

1. SILENT may also be specified as an option of the AT command in the following manner:

   ```
   [id] ATS ROUTINE10[;attachment] ...
   ```

   All attachments to a silent breakpoint function normally except MODIFY. The display of the old contents of the modified location is inhibited.

ACTIVE/INACTIVE Command

Format:

```
{AC[TIVE]  }  id [-id  ]
{IN[ACTIVE]}     [TO id]
```

Description:

These commands set stored commands to either an active or inactive status. An active breakpoint is one which is still in effect. An inactive breakpoint is one which is no longer in effect. An inactive breakpoint is remembered by DELTA, however, and will still maintain its identification number.

Example:

Command	Explanation
ACTIVE 10-15	Set the specified range of inactive breakpoints to an active status.
INACTIVE 10-15	Set the specified range of active breakpoints to an inactive status.

Usage Notes:

1. INACTIVE may also be specified as part of the AT command in the following manner:

   ```
   [id] ATI ROUTINE10[;attachment] ...
   ```

   Attachments to an inactive breakpoint will not function until the breakpoint is activated again, since an inactive breakpoint is never hit.

UPDATE Closed Form Command

Format:

```
UP[DATE]  id[ ,m]  text
```

Parameters:

id	is the identification of a stored command.
m	is the attachment number.
text	is the text to replace attachment m of stored command id.

Description:

The UPDATE command replaces attachment m of the specified stored command with replacement text. Attachment zero is the stored command. If attachment m does not exist an error message is given. If m is not specified, the replacement text is added as an attachment to the end of the specified stored command. If attachment m is replaced with all blanks, attachment m is deleted.

Example:

Command	Explanation
UPDATE 1,2 DI ALPHA	Replace the second attachment of stored command 1 with the text 'DI ALPHA'
UPDATE 3 PLUGH	Add the text 'PLUGH' as an attachment to the end of stored command 3.
UPDATE 2,0 AT PROG:100	Replace stored command 2 with the text 'AT PROG:100'

UPDATE Open Form Command

Format:

```
UP[DATE][A]  id[-id]
     or
UP[DATE]     id,m
```

Parameters:

id	is the identification of a stored command.
A	indicates that UPDATEing is to be done an attachment at a time.
m	is the attachment number.

Description:

The first format displays stored commands id to id, stopping at the end of each to allow modification of the stored command. If A is specified, for stored commands id to id, an attachment at a time will be printed, stopping at the end of each attachment to allow modification. Entering linefeed causes the next attachment or stored command to be displayed stopping at the end to allow modification. Entering ↑ causes the previous attachment or stored command to be displayed stopping at the end to allow modification. A carriage return ends updating on the current stored command. If a stored command is greater than 256 characters in length and A was not specified, a message will be given and the stored command will be updated one attachment at a time.

The second format will print attachment m of stored command id, stopping at the end to allow modification of that attachment.

Example:

Command	Explanation
UPDATE 1-4	Prints the stored commands in the range specified stopping at the end of each to allow modification.
UPDATEA 5-10	The stored commands in the range specified are printed an attachment at a time stopping at the end of each attachment to allow modification.
UPDATE 3,2	Prints the second attachment of stored command 3 stopping at the end to allow modification of that attachment.

SAVE Command

Format:

```
[id] SA[VE];attachment[;attachment] ...
```

Description:

This command is a keystroke saver. It saves attachable commands as though they were attached to a breakpoint. The SAVEed attachments may then be executed by referring to their id in a DO command.

DO Command

Format:

```
DO id
```

Parameters:

id	is the identifier of any stored command.

Description:

The DO command performs the attachments to any stored command. This includes attachments saved with the SAVE command and the attachments to any breakpoint command whether active or inactive.

Usage Notes:

1. The DO command is itself attachable, and allows an IF specification. This allows very complex, and possibly recursive, chains of attachments. The user must take care to avoid recursions which will never exit. The BREAK key can be used to stop an endless attachment loop.

Fault and Trap Control Commands

The fault and trap control commands include the following:

KEEP/TRAP/IGNORE, ON EXIT/ON ABORT

One of the functions of DELTA is to capture all faults, traps, asynchronous entries, normal exits, and abnormal exits which might occur during execution of your program. These are collectively referred to as exceptional conditions (ECs) by DELTA.

The action taken by DELTA when an EC occurs is controlled in part by the commands described in this category. The user has the choice of having DELTA report the type and position of the EC (KEEP), ignore the EC (IGNORE), or pass the EC to a handler in the user's program (TRAP). The user may also specify a list of attachments to be executed when an exit condition occurs (ON EXIT, ON ABORT).

KEEP/TRAP/IGNORE^Command

Format:

```
          {AL[L]                              }
          {N[ONE]                             }
          {category[,category] ...            }
          {type[,type] ...                    }
KE[EP]    {H[OST] AL[L]                       }
TRAP      {H[OST] N[ONE]                      }
IG[NORE]  {H[OST] type[,type] ...             }
          {fprg-res AL[L]                     }
          {fprg-res N[ONE]                    }
          {fprg-res type[,type] ...           }
```

Parameters:

ALL   specifies all categories and types are to be added to this list.

NONE   specifies all categories and types are to be removed from this list.

category   specifies a category to add to this list.
(Categories are in Table 3-3.
All types in a category may be placed on a list
by using the category name.)

type   specifies a type to add to this list.
(Types are listed in Table 3-3.
FEP exceptional condition types are listed in Table 3-4.)

FEP Parameters:

Host specifies the host program

fprg-res specifies a particular FEP program. This is the value the user specified for the RES on the M$OPEN of the FEP program.

Table 3-3. Exceptional Condition Types By Category

CATEGORY	EC TYPE	APPLICABLE MONITOR SVC
MO[NITOR]	T[IMER]	M$STIMER
	EV[ENT]	M$EVENT
	B[REAK]	M$INT
	X[CON]	M$XCON
	PM[ME] (Bad PMME, no ALTRET)	M$TRAP
AR[ITHMETIC]	OV[ERFLOW]	M$TRAP
	DI[VIDE_CHECK] (Divide check)	M$TRAP
PR[OGRAMMED]	MM[E]	M$TRAP
	DE[RAIL]	M$TRAP
	F[AULT_TAG]	M$TRAP
ER[ROR]	ME[MORY]	M$TRAP
	CO[MMAND]	M$TRAP
	L[OCKUP]	M$TRAP
	I[PR] (Illegal Procedure)	M$TRAP
	SEG (Missing Segment)	M$TRAP
	PA[GE] (Missing Page)	M$TRAP
	SEC_1 (Security 1)	M$TRAP
	SEC_2 (Security 2)	M$TRAP

Table 3-4. FEP Exceptional Condition Types

EC TYPE	APPLICABLE MONITOR SERVICE
EV[ENT]	M$EVENT
B[REAK]	M$INT
X[CON]	M$XCON
M[CL] (Bad MCL, no ALTRET)	M$TRAP
T[RAP]	M$TRAP

Description:

KEEP/TRAP/IGNORE are used to build a list of EC types which will be handled in one of three ways.

If an EC type is on the KEEP list when an EC of that type occurs, it is reported, along with the current position. If an EC is on the TRAP list when an EC of that type occurs, it will be passed to the user program's handler for that EC type. If an EC is on the IGNORE list when an EC of that type occurs, the program IC is adjusted to avoid re-occurrence of the EC and control returns to the program at the new position.

To remove an EC type from one list, the user specifies that it is to be placed on some other list. To remove all EC types from a list, the user specifies NONE for that list. Any EC types that were on a NONEed list are placed on the KEEP list.

Usage Notes:

1. Use of a category name implies all types in that category. Types and categories may be mixed freely on a line, e.g., TRAP MONITOR, DIVIDE.

2. If, at the time of an occurrence of an EC on the TRAP list, a program does not have a valid handler for that EC, an error message will be issued. DELTA will continue as if that EC had been on the KEEP list.

3. Precedence is given to ON EXIT/ON ABORT for the XCON EC.

4. When there are FEP programs associated, the commands have the following meanings:

   KEEP ALL   report everything on all programs - Host and FEP.

   KEEP HOST ALL   report everything on the Host program.

   KEEP fprg-res ALL   report everyting on the specified FEP program.

   KEEP HOST EVENT   report event exceptional conditions on the Host program.

   TRAP ALL   passes exceptional conditions on all programs to the program's trap handler - this does not include the start of FEP programs.

   TRAP HOST ALL   passes exceptional conditions on the Host program to the Host program's trap handler.

   TRAP fprg-res ALL   passes exceptional conditions on the specified FEP program to the specified FEP program's trap handler.

   TRAP fprg-res MCL   only pass MCL exceptional conditions for the specified FEP program to the specified FEP program's trap handler.

   IGNORE ALL   reports no exceptional conditions on all programs.

   IGNORE HOST ALL   reports no exceptional conditions on the Host program.

   IGNORE fprg-res ALL   reports no exceptional conditions on the specified FEP program.

   IGNORE HOST BREAK   reports no BREAK type exceptional conditions on the Host program.

ON EXIT/ON ABORT Command

Format:

```
[id] O[N] A[BORT] [condition][;attachment] ...
[id] O[N] E[XIT] [condition][;attachment] ...
```

Description:

These commands are primarily intended for use by the noninteractive user (batch mode or online !XEQ) who wishes to specify an action to be taken when his program aborts or exits unexpectedly. ON EXIT specifies a list of attachments to execute when an EXIT condition is encountered, ON ABORT specifies a list of attachments to execute on an abort condition.

An EXIT condition can be caused by M$EXIT, M$SAVE, and M$LDTRC. An abort condition can be caused by MERR, MXXX or a monitor abort for limit exceeded.

Example:

```
    ON ABORT;DU $TCB->0,64;Q
```

When an abort occurs, dump the top TCB frame, and then exit DELTA.

Usage Notes:

1. ON ABORT and ON EXIT are stored commands. They can be displayed and killed using SHOW id and KILL id.
2. ON ABORT and ON EXIT take precedence over KEEP/TRAP/IGNORE, i. e., even though TRAP XCON was specified, the attachments for ON ABORT or ON EXIT will be executed if an XCON EC (Exceptional Condition) occurs. If the user wishes to pass control to an XCON handler in his program after the attachments are executed, he attaches a GOTRAP command.
3. ON ABORT is not allowed for FEP programs.

Miscellaneous Housekeeping Commands

The miscellaneous housekeeping commands include the following:

BYPASS, PROTECT, SAD

This category of commands provides additional facilities that do not easily fall into any of the other categories of housekeeping commands. Commands are provided to:

1. Control stepping through GMAP6 modules.
2. Avoid inadvertent modification to memory locations.
3. Allow access via descriptors from the monitor's linkage segment.

BYPASS Command

Format:

BY[PASS]

Description:

This command is used to specify that DELTA is to skip step reporting while stepping assembly language program units. This is useful for PL-6 ECU's which call assembly language routines. When BYPASS is specified, all subsequent STEPs will not stop in assembly language modules, but will stop at the first instruction/statement after exiting from the bypassed module. BYPASS is the default unless the start address is in an assembly language routine; then the default is KILL BYPASS.

Usage Notes:

1. BYPASS is a toggle. It is reset with the BYPASS option of the KILL command.

PROTECT Command

Format:

PROT[ECT]

Description:

This command is used to specify that DELTA is _not_ to modify memory locations as specified by the MODIFY, LET, FIND, or STORE commands. The command is most often used when the interactive form of the MODIFY command is used to examine a run unit image in RUM mode, or when the running monitor is being examined in ANLZ mode, and the user wishes to insure that a typing error will not cause a memory location to be changed inadvertently. KILL PROTECT is the default.

Usage Notes:

1. PROTECT is a toggle. It is reset with the PROTECT option of the KILL command.

SAD Command

Format:

SAD n

Parameters:

n   is a decimal or octal number.

Description:

This command specifies which descriptor from the monitor's linkage segment is to be associated with the special symbol $SAD. (See the Monitor Services Reference Manual (CE33) M$SAD description for more information regarding special access descriptors.)

Usage Notes:

1. The SAD option of the SHOW command is used to display the currently associated descriptor number.

2. The SAD command is not allowed for FEP programs.

Section 4
Execution Control

This set of commands allows the user to specify that DELTA is to assume control of an executing run unit at any of the following times:

- When control passes to a procedure position (AT).
- When an elementary item or location is modified (WHEN).
- When the BREAK key is depressed.
- When one program unit CALLS another program unit (ON CALL).
- When an overlay node is loaded (ON NODE).
- When an exceptional condition occurs. (See the discussion on Fault and Trap Control Commands in Section 3.)

Once DELTA has assumed control and issued its prompt character, the user may then specify that execution is to resume at the same location; specify whether or not to proceed with execution by instruction or by statement; or issue any command in DELTA's repertoire.

Execution Control commands are divided into the following categories:

- Procedure breakpoint commands
- Data breakpoint command
- Transfer of control commands
- Procedure stepping commands
- Special purpose execution commands

Procedure Breakpoint Commands

Procedure breakpoint commands include the following:

AT, ON NODE/ON NODES, ON CALL/ON CALLS.

A procedure breakpoint occurs on the arrival of the Instruction Counter (IC) at a designated position (AT, ON CALL); on the general occurrence of CALLS (ON CALLS); or on the loading of an overlay node (ON NODE, ON NODES).

AT Command

Format:

```
[id] A[T][I][S] position [#n] [condition] [;attachment] ...
```

Parameters:

id   is a decimal number from 1 to 99 which identifies the AT command.

I   indicates that the breakpoint is to be inactive (see the INACTIVE command).

S   indicates that the breakpoint is to be silent (see the SILENT command).

#n   specifies the number of times to BYPASS the breakpoint with a true CONDITION before allowing it to be executed.

condition   is an IF attachment.

Description:

The AT command conditionally or unconditonally interrupts execution at a specific position in the logical flow of a run unit.

Example:

Command	Explanation
AT P:100	Sets a breakpoint at statement number 100 in ECU P.
AT P:100;DISPLAY X;GO	At P:100 the value of X is printed and execution is resumed.
10 AT P:100 IF X EQ 0;DISPLAY Y	The AT command has an id of 10. At P:100 if X equals 0, the value of Y is printed and DELTA reads its next command. If X is not equal to 0, execution resumes without interruption.
AT :7 #6; LET A 1; GO	Conditional execution is specified by using a bypass count (#6). The breakpoint will be bypassed six times, then reported and executed on the seventh time.

AT :10 IF A GT 20;
LET B 0 IF C > 10;
LET B 1 IF C LE 10;
GO THERE IF B EQ 1; GO
	Multiple levels of conditionality are specified using the IF conditional. Both the reporting of this breakpoint and the execution of its attachments are dependent upon the truth of the first IF conditional.
	When the first condition is true, the breakpoint is reported and the attachments are examined. The
	The attachments are executed only when their associated IF condition is true.

The previous example translates to the following pseudo code:

```
10:   IF A > 20
11:   THEN DO;
12:      IF C > 10 THEN B = 0;
13:      IF C <= 10 THEN B = 1;
14:      IF B = 1 THEN GOTO THERE;
15:      END;
```

Usage Notes:

1. When both BYPASS and IF are specified, the bypass count is decremented only when the IF condition is true. The breakpoint will be reported on occurrence n+1 of the true condition.

2. When the INACTIVE or SILENT options are specified, no space character is allowed between the AT and the option (e.g., ATS or AS means AT, SILENT).

3. All executable statements and static data are located in the Instruction Segment of a run unit. The AT command should always specify the address of an executable instruction. Because of the way that position may be specifed (symbols, octal addresses, decimal addresses, expressions), DELTA cannot guarantee that this requirement is met. DELTA places a breakpoint at the position to which the expression is resolved. Should this position not be an executable instruction, then the breakpoint will not occur and it is possible that the placement of the breakpoint may alter a word of data within the run unit. Should this happen, the results will be unpredictable. Special care should be exercised to avoid setting a breakpoint in the arguments of a calling sequence which are often intermingled with the executable instructions. Specification of a breakpoint using a program statement number or label will always perform correctly.

4. To continue execution from the point of interruption, use the GO command with no position specified. If you use GO position where position is the current IC, i.e., where the breakpoint occurred, the breakpoint will immediately re-occur. Any other execution resuming command can be used (that is, STEP, GOSTEP, XCON, BREAK, or GO position where position is not the current IC).

5. The same position may not be used in two different ATs at the same time; i.e.,

   ```
   3 AT 10;DISPLAY A;GO
   4 AT 10;DISPLAY B;GO
   ```

   is not valid.

ON NODE, ON NODES Commands

Format:

General Form:

[id] O[N] N[ODES] [condition][; attachment] ...

Specific Form:

[id] O[N] N[ODE] name [condition][; attachment] ...

Parameters:

id   is a decimal number from 1 to 99.

condition   is an IF conditional which must be true for the breakpoint to occur.

name   is the name of an overlay node.

Description:

The ON NODE command conditionally interrupts execution when an overlay is loaded.

Usage Notes:

1. When the general and a specific form of the command are active, they are mutually exclusive. The specific form is not activated unless the named overlay is loaded. The general form is not activated when any overlay named in a specific form of the command is loaded.

2. Only one general form is permitted, but many specific forms may be active.

3. The breakpoint occurs on the M$OLAY monitor service and $IC points to the instruction immediately after the CLIMB instruction.

4. The ON NODE(S) command is not allowed for FEP programs.

ON CALL, ON CALLS Commands

Format:

General Form:

```
[id] O[N] [X] C[ALLS] [condition][;attachment] ...
```

Specific Form:

```
[id] O[N] C[ALL] name [condition][;attachment] ...
```

Parameters:

id   is a decimal number from 1 to 99.

X   specifies breakpoints on external procedure calls only.

condition   is an IF conditional which must be true before the breakpoint is to occur.

name   is the name of a program unit.

Description:

The ON CALL command allows the user to conditionally interrupt execution prior to entry of a CALLed program unit.

Usage Notes:

1. When the general and a specific form of the command are active, they are mutually exclusive. The specific form is not activated unless the named program unit is CALLed. The general form is not activated when any program unit named in a specific form of the command is called.

2. Only one general form is permitted, but many specific forms may be active.

3. The breakpoint occurs before execution of the first instruction of the called program unit. The $IC points to the first instruction of the called program unit.

Data Breakpoint Command

WHEN is the data breakpoint command.

A data breakpoint occurs when the value of a variable changes during the execution of the program. DELTA can be told to stop when any change is made to the variable, or only when the new value fits a given relation with the specified constant.

WHEN Command

Format:

```
[id] W[HEN] item [relation constant] [#n]
    [condition][; attachment[; attachment] ... ]
```

Parameters:

id   is a decimal number from 1 to 99.

item   is an elementary item, structure or substructure, or single element of an array. A range of array subscripts may not be specified. The complete specification for item can be found in the section called Memory Display and Modification.

relation   any of the relations legal in the IF attachment described earlier ,e.g., =, >, <, GT, LE, etc.

constant   any of the literal constants known to DELTA, e.g., 7, 'ABCD', 2.2, '0'B, .134, etc.

#n   specifies the number of times to bypass the breakpoint with a true condition before allowing it to be executed.

condition   an IF attachment.

Description:

This stored command causes data breakpoints to occur when the value of a named variable or structure changes.

Example:

In the examples in this subsection, assume the following program section is defined:

```
    DCL 1 LEN_REC STATIC,
          2 NAME,
            3 LN CHAR(12),
            3 FN CHAR(10),
            3 MI CHAR(1),
          2 SSN CHAR(9);
```

Command	Explanation

WHEN SSN	Halt execution when the value of SSN changes from the value it had when the command was first issued. Both of the following statements could cause the example data breakpoint to be hit:
	
	SSN = ' ';
	LEN_REC = IN_REC;

WHEN FN EQ '**'	Halt execution when the value of FN is changed to '**'. Note that if FN is equal to '**' when the command is issued, the data breakpoint will not be hit until FN is changed to something other than '**', and then back to '**'.
WHEN SSN='999999999'; DISPLAY NAME;GO	This example shows that attachments can be made to WHEN in the same manner as the AT command. The IF conditional may also be used.

Usage Notes:

1. If more than one data breakpoint variable is changed at the same time, only the breakpoint with the smallest ID is considered to be hit. For example, consider the following two breakpoints, using the structure defined above:

   ```
   4  WHEN NAME
   5  WHEN SSN
   ```

   The statement MOVE SPACE TO LEN_REC would cause only breakpoint 4 to be hit, and not 5, even though the value of SSN may have changed. If NAME was already equal to space, but SSN was not, breakpoint 5 would be hit in this example, since the value of NAME did not change.

2. While duplicate positions are not legal for multiple AT commands, duplicate variables may be specified for WHEN, e.g.,

   ```
   WHEN A>10
   WHEN A<4
   ```

   can exist together. If two or more data breakpoints are hit at the same time (overlapping conditions or data), only the first (smallest id) will be honored.

3. If a variable is used as a subscript for an array, the value of the variable subscript is stored in the WHEN command, and not the variable. For example, assume the value of A is 7. WHEN ARRAY(A) is stored as WHEN ARRAY(7) by DELTA.

4. Data breakpoints are very costly in terms of computer resources. They will greatly increase the execution time of the program, more so than any other DELTA command. DELTA works with the CP-6 monitor to watch the variables in the program change in value. The smallest area of memory that can be watched is a page (that is 1,024 words) of memory. This means that if WHEN A is specified, DELTA is notified anytime anything on the page or pages containing A is modified. DELTA then checks to see if the value of A has changed. If all of the variables in the program are stored in the same page of memory, DELTA is entered every time any variable is modified, or just about every statement.

   Always try to localize the area of the program that is to monitor a variable. For example, assume you, the user, are only interested in changes made to A between statements 100 and 150. The following method can be used to reduce the amount of overhead used by DELTA.

>10 WHEN A	Defines the breakpoint assigned with a number of 10.
>INACTIVE 10	Turns it off.
>ATS 100; ACTIVE 10;GO	Silently turns it on at 100.
>ATS 150; INACTIVE 10;GO	Silently turns it off at 150.

   AT is much less costly than having a data breakpoint in effect all the time. After the WHEN has served its purpose, KILL it.

5. The WHEN command is not allowed for FEP programs.

Transfer of Control Commands

The transfer of control commands include the following:

GO, BREAK, GOTRAP.

The commands in this category allow the user to conditionally or unconditionally alter the flow of program control.

GO Command

Format:

```
     [position              ]
G[O] [AL[L]                 ]
     [H[OST] [position]     ]
     [fprg-res [position]]  ]
```

Parameters:

position   will cause execution to resume at the specified position. If not specified, position is assumed to be the current value of $IC.

FEP Parameters:

ALL   causes program execution to resume at the current position for all programs — Host and FEP.

HOST   causes program execution to resume either at the current position or at a specified position for the Host program.

fprg-res   causes program execution to resume either at the current position or at a specified position for the specified FEP program.

Description:

This command is used to resume program execution either at the current position or at a specified position.

Example:

Command	Explanation
GO 100	Resume execution at statement number 100 of the current ECU.
GO READ_REC	Resume execution of the statement labeled READ_REC in the current ECU.
GO ECUB:100	Resume execution at statement number 100 of ECUB.
AT 100;DISPLAY X; GO	When control reaches statement 100 of the current ECU, display X and continue execution.
AT 100 IF A > B ;DISPLAY X; GO 200 IF A EQ 40; GO	When control reaches statement number 100 of the current ECU, if A is greater than B, then display X and continue execution at 200 if A is equal to 40 or at 100 if A is not equal to 40.

Usage Notes:

1. Users of higher level language processors are warned that some language compilers generate optimized code. This means that some statements are combined by the compiler, and cannot be executed out of order if the proper results are to be obtained. For example:

```
    19:   B = 6
    20:   A = B+5+D
    21:   C = B+5+E
```

   The compiler might compute the value of B+5 in statement 20 and save it in a register of temporary storage location. For statement 21, the value of B+5 is fetched from the saved cell, and used in the equation. B+5 is not recomputed. This saves time and reduces the amount of code produced. However, if a breakpoint is placed at 20, and a GO 21 is performed, statement 20 is never executed. The value B+5 is never computed and a possibly wrong answer will be produced by statement 21.

   This example is simple, but shows that the GO [position] command should be used with care in code produced by optimizing compilers. In general, at least for PL-6, FORTRAN and COBOL, it is safe to "GO label" (paragraph name for COBOL). The compiler is expecting multiple entry paths to that statement, and will not generate code that relies on previous statements.

BREAK Command

Format:

B[REAK]

Description:

The BREAK command causes control to be given to the break handler (declared in an M$INT monitor service call). A BREAK frame is placed on the user Task Control Block (TCB) before giving control to the break handler.

Usage Notes:

1. The BREAK command should not be confused with the break key. Depressing the break key at any time during a debugging session gives control to DELTA unless the user has specified TRAP BREAK. If TRAP BREAK has been specified, DELTA gives control to the break handler without reporting the break. To direct a break to DELTA even though TRAP BREAK has been specified, the user enters CTRL Y, then DELTA. This will simulate a break that DELTA will not pass to the break handler.

GOTRAP Command

Format:

```
             [position                ]
G[O]T[RAP]   [H[OST] [position]       ]
             [fprg-res [position]]
```

Parameters:

position   will cause execution to resume at the specified position.

FEP Parameters:

HOST   causes control to be transfered to the Host program's exceptional conditions routine for the condition that caused entry to DELTA.

fprg-res   causes control to be transferred to the specified FEP program's exceptional condition routine for the condition that caused entry to DELTA.

Description:

This command transfers control to the exceptional condition routine for the condition that caused entry to DELTA. (This command is used to enter the TRAP routine and the EVENT routine.)

Usage Notes:

1. When the command is issued, the information about the exceptional condition is moved into place in the TCB. If position is specified, the Instruction Counter (IC) value in the TCB is set to reflect that position. Execution resumes at the handler for the exceptional condition that caused entry to DELTA.

2. For FEP programs:

   If position is specified, the Instruction Counter (IC) value in the TSA is set to reflect that position. Execution resumes at the specified handler for the exceptional condition that caused the specified FEP program's entry to DELTA.

Procedure Stepping Commands

The procedure stepping commands include the following:

    STEP, GOSTEP, GOTRAPSTEP.

The commands in this category allow continued execution one unit at a time. Possible unit sizes are INSTRUCTION, SUBSTATEMENT, STATEMENT, and PARAGRAPH. ONE CALL is a special case unit.

STEP Command

Format:

S[TEP] [n]

Parameters:

n   is an octal or decimal constant specifying the number of steps to be taken. If n is omitted, one step is taken. The right bracket (]) may be used in place of STEP. The STEP command format then becomes] or n].

Description:

The STEP command causes execution to proceed for a given number of steps.

Example:

Command	Explanation
STEP 5	Take 5 steps (instructions or statements, depending on the context in which the command is used).

STEP BY Command

Format:

S[TEP] B[Y] mode

Parameters:

mode is one of the following:

Mode	Explanation
I[NSTRUCTION]	A machine level instruction.
SU[BSTATEMENT]	A substatement as defined by the compiler. A substatement is usually defined as a single statement in a multiple statement line; that is, a single source code line in which more than one statement appears.
ST[ATEMENT]	A source statement. (The default step mode is STATEMENT.)
P[ARAGRAPH]	A COBOL PARAGRAPH, or the next source line containing a label in non-COBOL ECUs.

Description:

The STEP BY command sets a new step mode (unit size), and then steps once.

Example:

Command	Explanation
SBI	Step one instruction and set the step mode to INSTRUCTION.
SBST	Step to the beginning of the next statement, and set the step mode to STATEMENT.

Usage Notes:

1. Use SHOW STEP to display the current step mode.

STEP ONE CALL Command

Format:

S[TEP] O[NE] C[ALL]

Description:

This command temporarily overrides the current step mode without setting a new one. The STEP O[NE] C[ALL] is used only when the current user program statement is a CALL statement. Execution will next stop at the statement immediately following the CALL statement, or at the ALTRET statement, if the called routine ALTRETURNs. The current mode is unchanged.

Example:

Command	Explanation
SOC	DELTA will halt execution when the CALLed routine RETURNS or ALTRETURNs.

Usage Notes:

1. This command is used to continue stepping in the current routine, without stepping through a called routine.

GOSTEP Command

Format:

```
             [position               ]
G[O]S[TEP]   [H[OST] [position]      ]
             [fprg-res [position]    ]
```

Parameters:

position   will cause execution to resume at the specified position.

FEP Parameters:

HOST   will cause execution to resume at the current position or specified position in the Host program.

fprg-res   will cause execution to resume at the current position or at the specified position in the specified FEP program.

Description:

This command performs the GO function but, at the same time, directs that a single step (instruction or statement) be performed.

Usage Notes:

1. One step will be performed at position. This command is equivalent to:

```
LET $IC position
STEP
```

GOTRAPSTEP Command

Format:

```
                [position              ]
G[O]T[RAP]ST[EP] [H[OST] [position]     ]
                [fprg-res [position]]
```

Parameters:

position   will cause execution to resume at the specified position.

FEP Parameters:

HOST   will cause execution to resume at the current position or the specified position in the Host program.

fprg-res   will cause execution to resume at the current position or the specified position in the specified FEP program.

Description:

This command performs the same function as the GOTRAP command except that the step level of operation is placed into effect.

Usage Notes:

1. One step is performed from the beginning of the exceptional condition handler for the condition which caused entry to DELTA. If position is specified, the IC value in the TCB frame for the exceptional condition is set to reflect that position.

2. For FEP programs:

   If position is specified, the IC value in the TSA frame for the exceptional condition is set to reflect that position.

Special Purpose Execution Commands

The special purpose execution commands include the following:

XCON, EXIT, ALIB, UNSHARE.

The commands in this category allow interaction with the monitor service calls M$XCON, M$SAVE, M$LDTRC, M$LINK, M$ALIB, and M$UNSHARE. (Knowledge of the appropriate areas of the Monitor Services Reference Manual (CE33), especially the section on Exception Condition Processing, is assumed in the following discussion.)

XCON Command

Format:

```
XC[ON]
```

Description:

This command simulates an exit control condition, that is, a condition which will cause entry to the routine specified in an M$XCON monitor service call. It is most useful in avoiding the one condition where having DELTA associated with a run unit will cause different results.

Assume a run unit has an exit control (M$XCON), but not a trap control (M$TRAP) routine. When the program runs without DELTA, any error or abort condition will cause direct entry to the exit control routine. However, when the program runs under the control of DELTA, these conditions are caught and held by DELTA, and reported to the user. In this case, the GOTRAP command will not work: DELTA checks to ensure that the program has a trap routine for the error or abort condition. The XCON command must be used to give control to the exit control routine.

The XCON command may also be used to give control to an exit control routine for exit conditions such as M$EXIT, M$ERR, M$XXX, and M$SAVE or M$LDTRC. In these cases, GOTRAP will work, since DELTA treats the exit control routine specified by M$XCON as a trap control routine for exit conditions. The TRAP XCON command will also work for exit conditions, for the same reason.

EXIT Command

Format:

```
EX[IT]
```

Description:

DELTA checks to ensure that the user does not force an exit condition while the run unit is already in exit control, as this will cause the program to be run down, and control to return to IBEX. This feature makes it difficult for a user to terminate a debug session unintentionally. However, there are three conditions which require an exit from exit control to continue debugging: M$SAVE, M$LDTRC, and exit from an M$LINKed-to program. The EXIT command is used to continue an M$SAVE or M$LDTRC after the final exit from exit control. In an M$LINKed-to program, EXIT is used within the exit control routine, or after the first exit if the program has no exit control routine.

Usage Notes:

1. The EXIT command is not allowed for FEP programs.

ALIB Command

Format:

```
ALI[B] {R[ETURN]    } [, K[EEP]]
       {A[LTRETURN]}
```

Parameters:

RETURN	causes a RETURN to the M$ALIB call.
ALTRETURN	causes an ALTRETURN to the M$ALIB call.
KEEP	overrides the DLIB option on the M$ALIB call, forcing DLIB=NO and directs DELTA to detect faults, etc.

Description:

The ALIB command causes a RETURN, or an ALTRETURN if DELTA is entered from the M$ALIB monitor service call. The GO command may also be used, except if DELTA has been placed in the ANLZ or RUM mode.

Usage Notes:

1. The KEEP option is not legal when M$ALIB is issued from the ASL domain.
2. If the RETURN=YES option is used, an ALIB RETURN is performed by DELTA after the command specified by the CMD option is performed.
3. If the M$ALIB call does not specify ALTRET, ALIB ALTRETURN is equivalent to ALIB RETURN.
4. The ALIB command is not allowed for FEP programs.

UNSHARE Command

Format:

```
         [A[LL]
UNSHARE  [P[ROGRAM]]
         [L[IBRARY]]
```

Parameters:

ALL   specifies both the program and library. PROGRAM is the default.

Description:

The UNSHARE command causes the program and/or library of an autoshared run unit to be unshared via the M$UNSHARE monitor service call.

Section 5
Execution Tracing

The commands in this category cause the flow of control within a run unit to be recorded and displayed upon command. A HISTORY mode may be set such that trace information is saved for examination at a later time. Tracing may be conducted at several levels:

- On all entry points
- On specific entry points
- On a specific entry point
- On all transfer points
- On specific locations

DELTA maintains a circular history buffer in which data pertaining to each incident causing a change in the flow of control is stored.

The commands described in this section are discussed in the following order:

```
    TRACE
    HISTORY
    PLUGH
```

TRACE Command

Format:

```
T[RACE] {[X] C[ALLS][S] [prefix]}
        {transfer-type[S]       }
        {O[N]                   }
        {OF[F]                  }
```

Parameters:

X   specifies that only external calls are to be traced.

prefix   specifies that only calls to procedures whose name begins with the character string defined by prefix are to be traced.

OFF   turns current TRACE mode off.

ON   turns current TRACE mode on.

S   specifies tracing is to be silent, i.e., entries are to be made to the history buffer, but are not displayed. They may be displayed later with the HISTORY command.

Table 5–1. TRACE Transfer Types

Transfer Type	Description
T[RANSFERS]	Traces all TSX instructions.
S[TATEMENTS]	Traces branches to the first instruction of a statement.
P[ARAGRAPHS]	Traces branches to the first instruction of a COBOL paragraph.

Description:

The TRACE command is used to invoke the maintenance of a history buffer by DELTA. The contents of the buffer are dependent upon the type or types of trace specified by the various options of the commands.

Example:

Command	Explanation
TRACE X CALLS	Trace calls to all external procedures.
TRACE CALLSS	Trace all calls, silently.
TRACE CALLS S	Trace all calls to procedures whose names begin with S.
TRACE TRANSFERS	Trace all TSX (transfer) instructions.
AT FMN$OPNF:100;TRACE CALLS;GO	When control is received at FMN$OPNF:100, activate the TRACE CALL mode and proceed.

Usage Notes:

1. SHOW TRACE displays the current TRACE mode, and KILL TRACE is used to remove the current TRACE mode.

HISTORY Command

Format:

```
H[ISTORY] [n [M[ORE]]]
          [C[LEAR]    ]
```

Parameters:

n is a decimal integer which specifies the number of history entries to be displayed. If n is not specified, all entries, or remaining entries, will be displayed.

MORE is a keyword which specifies that the display is to continue from the last entry displayed.

CLEAR removes all entries from the buffer.

Description:

When one of the TRACE modes is active, DELTA maintains a list of the trace reports in the history buffer. The history buffer can hold 100 entries, and is circular, meaning the 101st entry will replace the first entry; the second is replaced by the 102nd entry. The buffer is also last-in, first-out, meaning that the latest (most recent) buffer entry is the first to be displayed. Thus, HISTORY shows a backwards flow of control.

Example:

Command	Explanation
HISTORY 3	Display three history buffer entries starting with the latest entry.
H3M	Display the next three history buffer entries.

Example of TRACE and HISTORY:

```
>TRACE CALLS
>G
A:3<CALLED> B:1
B:15<CALLED> C:1
C:120<CALLED> G:1
BREAK @ G:27   (User hits break)
>HISTORY 2
C:120<CALLED> G:1
B:15<CALLED> C:1
>HISTORY 5 MORE
A:3<CALLED> B:1
No more history
>
```

Usage Notes:

1. The HISTORY command is used in conjunction with the TRACE command. If the command is issued before one of the TRACE commands has been entered, the history buffer will contain no entries.

2. The history buffer is cleared (reset to null) whenever a KILL TRACE command is issued.

PLUGH Command

Format:

PL[UGH] [location]

Parameters:

Location may be:

- o an octal or decimal literal
- o an address expression (location reference or position reference)
- o a symbolic name

Description:

The PLUGH (Procedure List Used to Get Here) command displays the chain of procedure calls which led to the arrival of the Instruction Counter at the current point. The chain links are displayed on a last-in, first-out basis. DELTA's source for the procedure list is the user's automatic stack. A line is displayed which identifies the CALL statement which caused the entry to each currently active procedure. If location is specified, DELTA uses the location as the base of the automatic stack.

Example:

Assume 3 ECU's, which call each other in this order:

```
Y2 calls ROOM
ROOM calls WALL
WALL calls BUILDING

>GO
IPR FAULT ● BUILDING:15,,.1   (a program bug is hit.)
>PLUGH

WALL:93/TSX1 BUILDING:3       (PLUGH shows how the program
                              arrived at BUILDING from Y2.)
ROOM:70/TSX1 WALL:1
Y2:31/TSX1 ROOM:2
```

Assume Z calls X
X calls A

```
>REPORT O
>PLUGH

X+.262/TSX1 A(+.60)
Z+.360/TSX1 X(+.32)
Z/TSX0! X66_ARET+.1036 (X66_MAUTO)(+.4)
Bottom Frame
```

Usage Notes:

1. The command is useful only in those run units with activation stack frames.
2. If the REPORT command is issued with the OFFSET option before doing a PLUGH, the PLUGH command will display the offset into the user's automatic stack for each procedure. If the offsets are not preceded by a "+", the offsets are actual addresses of static parameter blocks.
3. A system call to an asynchronous entry will be indicated as a Bottom Frame.

Section 6

Memory Display and Modification

The commands in this category display and change the contents of both memory and program visible registers. The specification of an area of memory is subject to the rules detailed in Section 2. The specification of a program visible register is made using the appropriate DELTA distinguished name as listed in Appendix A.

The memory display and modification commands are divided into the following categories:

- o Variable oriented commands
- o Word oriented commands

Variable Oriented Commands

The variable oriented commands include the following:

DISPLAY, LET

The DISPLAY and LET commands are convenient means of displaying and modifying program variables. When the debug schema is available to DELTA, the user need not be concerned with specifying any of the attributes of the symbolic name referenced by either of these commands. Although DISPLAY and LET are intended for use with variable names, no restriction exists concerning the use of address expressions.

DISPLAY Command

Format:

```
D[ISPLAY] item[\f][,item[\f] ... ]
```

Parameters:

item   may be:

- o A scalar variable
- o A major structure
- o A qualified subordinate structure
- o A qualified elementary item
- o An array
- o A subscripted element of an array
- o A subscripted range of an array
- o Any general location reference (address expression)
- o Any general position reference
- o A DELTA distinguished name

f   is one of the format specifiers from Table 3-1 (see the FORMAT command). In general, specification of a format for variables is not necessary since each variable is described in the debug schema allowing DELTA to display it in the correct form. General address expression items are displayed in octal by default, while general position references are displayed in instruction format. The default display mode for any item may be overridden by use of the desired format specifier.

Description:

The DISPLAY command is used to display an area of memory or a program visible register.

The DISPLAY command displays program variables by name. The user need have no knowledge of their location or internal representation. DISPLAY handles simple variables, structured variables, arrays and all of the storage classifications used by the supported languages. These are static, automatic, parameters (both in automatic stack frames and in static storage) and based for PL-6; File Section, Linkage Section and Working Storage Section in COBOL; and all FORTRAN variables including subroutine parameters. The allowable forms of reference are as varied as the different data types. The PL-6 pointer variables in a pointer qualified reference may be part of a structure and may be subscripted. Subscripts themselves may be pointer qualified and/or subscripted.

Example:

Command	Explanation
DISPLAY HAROLD	Simple elementary variable.
DISPLAY JOHN(2)	Subscripted variable, using literal subscript.
DISPLAY JEFF(HAROLD)	Subscripted variable using elementary variable for the subscripted value. Note: A variable used for a subscript must be a signed or unsigned integer (PL-6, FORTRAN, COBOL) or a packed or unpacked decimal number which represents an integer value (COBOL).
DISPLAY PVAR->BASEDITEM	Pointer qualified variable.
DISPLAY SPVAR(3)->BASEDITEM	Pointer qualified variable using subscripted pointer variable.
DISPLAY SPVAR(PVAR->BASEDITEM)-> OTHERBASE(PVAR->NEWVAL)->MYBASEDVAL	General pointer qualified subscripted reference.
DISPLAY ALPHA.BETA.GAMMA	Structure qualified item. Note: All levels of a structure must be specified.

A subset of an array may be displayed by specifying a range of subscripts for one of its dimensions. A range is specified by two subscript values separated by a colon (:). If a subscript or subscript range is not specified for an array item, the entire array will be displayed. For example:

```
DISPLAY VECTOR(FIRST:LAST)

DISPLAY MATRIX(3:LASTITEM,THISROW)

DISPLAY VECTOR
```

Usage Notes:

1. PL-6 based variables declared with implicit pointers may be displayed without specifying the implied pointer, or explicit pointer qualification may be given to override the default implication.

2. While a variable to be displayed may have a range of subscripts specified, a subscripted pointer used in a pointer qualified reference may not. That is:

```
    DISPLAY PTR(2)->VAR.X      is allowed

    DISPLAY PTR(2:5)->VAR.X    is not allowed.
```

3. To DISPLAY a variable not described (referenced) in the current program unit requires procedure qualification in order to indicate to DELTA where the schema describing the item may be found. Procedure qualification is specified by giving the external program unit name of the EPU containing the variable followed by a colon (:). For example:

```
    DISPLAY SUBPROGRAM1:XVAR

    DISPLAY MAIN:LOOPCOUNT
```

   In PL-6, which has internally nested procedures (IPU), a variable which is local to an IPU must be qualified by the EPU and all levels of IPU to that containing the variable.

   Assume an external PL-6 procedure called EXPROC with internal procedure LOCPROC which, in turn, contains the procedure to ALPHA. To display a variable DATES which is local to ALPHA when the program counter is not currently within ALPHA, the reference is:

```
    DISPLAY EXPROC:LOCPROC:ALPHA:DATE
```

4. Only one procedure qualification specification is allowed for each item to be displayed. This means that all pointers and subscript variables in a general variable reference must be in the same procedure. For exceptions to this rule see the discussion of the ALTERNATE VARIABLES command in Section 3.

5. The DISPLAY command also allows the display of addressed locations of memory and the program-visible registers. In pointer qualified references it is allowable to use general address expressions for specifying the pointer location together with a based variable name for the item to be displayed. The DELTA distinguished names for the program visible registers are listed in Appendix A. For example:

```
    DISPLAY .400

    DISPLAY .41235->GORGO.PUUDLY

    DISPLAY $LS1->B$JIT.UNAME

    DISPLAY $X1,$Q
```

6. General address expressions are further discussed later in this section under Word Oriented Commands.

LET Command:

Format:

L[ET] item constant

Parameters:

item	may be as described under DISPLAY.
constant	is any meaningful literal in the current syntax (as specified by the currently specified SYNTAX option. (See the SYNTAX command in Section 3.)

Description:

The LET command is used to modify the contents of an area of memory or a program visible register.

Some literals are allowed and mean the same regardless of the current syntax. They are:

o	bit string '0010101'B	up to 72 bits - left justified
o	octal bit string '123'O	up to 24 octal digits - left justified
o	octal value O'123'	up to 12 octal digits - right justified
o	hex value X'AB29'	up to 9 hex digits - right justified
o	character string 'ABC' or 'ABC'C or C'ABC'	up to 63 characters - left justified

o a pointer literal of the form:

```
    octal word offset [-char offset[-bit offset]],segid
```

where segid is of the form:

```
    $LSn, $ASn, or $PSn (n is a decimal number)
```

For example:

```
    .123,$LS0
    .212-0-4,$LS4
```

Numeric literals are treated differently depending on the SYNTAX option which is in effect. This allows numbers to be specified in the form which is natural to the language being debugged.

When SYNTAX FORTRAN is in effect, integer and real values may be specified. Integer to real, and real to integer conversion is performed as necessary when the format of a numeric value conflicts with the variable data type. Real value constants are specified the same as for the FORTRAN INPUT statement except that blanks are not allowed. For example:

```
4.5, .2, 0.13E-8, 2D12, 6
```

When SYNTAX COBOL is in effect, integer (COMP-6) and decimal (COMP) values may be specified. Truncation or scaling takes place as necessary to match the input value with the scaling specified for the variable. The user is not required to distinguish between integer and non-integer values.

When neither SYNTAX COBOL nor SYNTAX FORTRAN is in effect, the period (.) is used to denote octal values, right justified in 36 bits. Numbers not preceded by a period are assumed to be binary integers. For example:

3,4,1269	decimal specification of binary integer
.4,.123	octal specification of binary integer
1.3	not legal
.129	not legal

Variable references for the LET command are exactly the same as for the DISPLAY command.

A blank is required between item and literal. This blank is interpreted as the equal sign (=). The use of the = sign in place of the blank is allowed if DELTA's end-of-message activation set is turned off with the KILL EOM command.
Usage Notes:

1. If the literal is a character string (e.g., 'ANDERSON'), then the variable must be a character string, or any item whose length in bits is 0 modulo 9 (an integral number of bytes). The character string constant can be up to 63 characters and is stored in the variable left-justified with truncation or blank-filled on the right as necessary to match the defined size of the variable.

2. If the literal is a pointer literal, the variable must be a pointer or any item whose size is 36 bits.

3. If the constant is a bit string or octal bit string, it is stored without regard to variable type, left-justified with truncation or zero-filled on the right as necessary to match the defined size of the item.

4. If the literal is an octal or decimal value, it is treated as a 36-bit constant and can be stored in any 36-bit variable or in any integer variable regardless of size, right-justified with truncation or zero-filled on the left as necessary to match the size of the variable.

5. When specifying item with an expression (e.g., simple octal addresses) or any variable or SYMDEF name with an offset expression, the location to which the expression resolves is treated as a 36-bit variable.

Cross Reference:

See the SYNTAX command.

Word Oriented Commands

The word oriented commands include the following:

EVALUATE, MODIFY, DUMP, FIND, STORE, PMD

These commands search, display, and modify memory. The user refers to memory locations as 36-bit words, that is, as a memory cell. The MODIFY command always refers to a single cell, while the DUMP, FIND and STORE commands may reference a single cell or a contiguous range of cells. EVALUATE is used to evaluate the address of a variable or general address expression and to perform miscellaneous arithmetic conversions.

DELTA allows considerable latitude in the methods by which the desired cell or cells may be designated. These are:

- Any octal or decimal literal.

- Any valid position or location reference.

- Any general variable reference. Note, however, that inasmuch as program variables are not necessarily word oriented, DELTA evaluates a general variable reference to the address of the word containing the start of the referenced variable. This means that if THREEBIT is a bit string variable of length 3 bits, starting with bit 10 of cell .2000, any reference to THREEBIT with a word oriented command such as MODIFY will cause the entire 36-bit contents of cell .2000 to be displayed and/or modified.

EVALUATE Command

Format:

E[VALUATE] expression[\f]

Parameters:

expression   may be:

- o A symbolic name (e.g., EVALUATE SYMBOL).
- o A symbolic name plus offset (EVALUATE SYMBOL+.10).
- o Simple octal or decimal addresses (EVALUATE 1329\O).
- o Simple address expressions with plus (+), minus(-), shift left (<<), and shift right (>>). (EVALUATE .60630>>2+3).
- o Simple pointer-qualified expressions using SYMDEFS or ENTDEFS even when schema usage is disabled. (EVALUATE B$JIT$->.12).
- o All forms of pointer and descriptor qualified references as discussed in Section 2.

  (EVALUATE $LS2->MYSTRUCT.NEXT->MS.SS.ITEM).

- o Any expression that can be resolved and expressed within a 36-bit word. This includes:

  Binary values (signed or unsigned). Bit strings up to 36 bits. Character strings up to 4, 9-bit bytes. GMAP6 assembler instructions.

- o If a format is specified, it does not change the default format for the MODIFY command. f is one of the format specifiers from Table 3-1.

Description:

This command is used to:

- o Perform simple arithmetic.
- o Reduce an address expression to its simplest form.
- o Obtain an address expression for use in some word oriented command.
- o Relate memory addresses to symbolic locations and positions within a run unit.
- o Convert between numbering systems.

Example:

Command	Explanation
EVALUATE 35+.13\U = 46	Do arithmetic with mixed numbering systems.
EVALUATE .40<<18 = .40000000	Perform logical arithmetic.
EVALUATE $LS5->MYSTRUCT.NEXT->MS .3712	Reduce this expression to its simplest form.

```
EVALUATE .2011\R                    To what part of my run unit
= PROG3+.5 :12(LOOP)                does word .2011 (of the IS)
                                    relate?

EVALUATE PROG3                      Display the octal offset of
= .2004                             PROG3 within the IS.

EVALUATE J                          What expression may I use to
= $LS4->.7                          address the symbol J which is
                                    not contained within the IS?

EVALUATE LDQ P3DEF+.12,X3,PR2       What is the octal representation
= .200422236113                     of this symbolic GMAP6
                                    instruction?
```

Usage Notes:

1. There is no operator precedence; operations are performed left to right.
2. Default format for output of EVALUATE is octal unless changed by the FORMAT EVALUATE command.
3. Format specification is ignored if the resultant address is not in the Instruction Segment (IS), that is, cannot be expressed as a value. When this occurs, the output is always segid->octal offset (e.g., $LS6->.1236).
4. The EOM (activation) character '=' can be used in place of EVALUATE. The format is:

   ```
   expression[\f]=
   ```

   For example:

   ```
   35+.13\U=46  (The 46 is supplied by DELTA).
   ```

Cross Reference:

See the FORMAT command.

MODIFY Open Form Command

Format:

```
M[ODIFY] location[\f][/]
```

Parameters:

location may be:

- o an octal or decimal literal
- o an address expression (location reference or position reference).
- o a symbolic name.

f may be any valid format specifier.

/ is the activation form of MODIFY and is allowed if KILL EOM has not been specified.

Description:

When the open form of MODIFY is used, DELTA first displays the address of the cell to be modified followed by its current contents and then prompts for the value to which the cell is to be modified. The prompt character (>) is preceded by the letter M (M>) which indicates that DELTA is in the modify-active mode. This means that:

o The indicated address is open for modification.

o DELTA expects either:

  - A new value to be placed in the open cell and the cell to be closed.

  - A new value to be placed in the open cell, the open cell to be closed and another cell to be displayed and opened for modification as specified by one of the EOM characters or MODIFY sub-commands. See End-of-Message Characters (EOM).

  - The open cell to be closed without modification.

End-of-Message Characters (EOM)

Several special EOM characters (in addition to /) apply to the open form of the MODIFY command. If the EOM activation character set has been disabled with the KILL EOM command, a special set of sub-commands is provided to indicate the same activity. The EOM activation characters along with their substitute sub-commands are shown in Table 6-1.

Table 6-1. EOM Characters and Sub-Commands

Character	Sub-Command	Action Indicated
Linefeed	N[EXT]	Open next memory cell for modify. Cell will be opened and displayed as if it has been addressed directly with a MODIFY command.
Up Arrow	P[REV]	Open previous cell for modify. The cell preceding the current cell will be opened and displayed as if it has been addressed directly with a MODIFY command.
Left Bracket Left Brace	O[PEN]	Re-open and re-display the last cell addressed by a MODIFY command. DELTA remembers the address of the last cell (if any) referenced in a MODIFY command.
Tab Character	Asterisk (*)	Indirect addressing. Display and open for modification the cell specified by the contents of the currently open cell. The interpretation of the address contained within the cell is dependent upon the format of the current display. If the current display is in pointer format, then the "segid" portion of the pointer is used to determine the appropriate segment, and the word offset portion of the pointer determines the

Table 6-1. EOM Characters and Sub-Commands (cont.)

Character	Sub-Command	Action Indicated
		offset within the segment. If the current display is in relative format, then the right half (the least significant eighteen bits) of the currently open cell is assumed to specify an address in the instruction segment. If the current display is in other than pointer format, then the left half (the most significant eighteen bits) of the currently open cell is assumed to specify an address in the instruction segment.
[\f]/	None	Same as * except do not open the call for modification. Optionally a format specifier may be used.
None	*L[\f]	Treat left 18 bits as an address regardless of displayed format. Optionally a format specifier may be used.
None	*R[\f]	Same as *L except that the right half (the least significant eighteen bits) of the currently open cell is assumed to specify an address in the instruction segment. Optionally a format specifier may be used.
None	*P[\f]	Treat contents of the currently open cell as a pointer regardless of the format in which it was displayed. Take same action as for the tab EOM character or the * sub-command. Optionally a format specifier may be used.
FEP EOM Characters:		
None	*L[\f]	Invalid for FEP programs.
None	*R[\f]	Invalid for FEP programs.
None	*P[\f]	Treats the contents of the currently open cell (2 words) as a pointer. Take the same action as for the tab EOM character or the * subcommand. Optionally a format specifier may be used.
None	*SP[\f]	Treats the contents of the currently open cell (1 word) as a 16 bit address. Take the same action as for the tab EOM character or the * subcommand. Optionally a format specifier may be used.

Table 6-1. EOM Characters and Sub-Commands (cont.)

Character	Sub-Command	Action Indicated
Tab Character	Asterisk (*)	Indirect Addressing. Display and open for modification the cell(s) starting at the address specified by the contents of the currently open cell(s). The interpretation of the address contained within the cell(s) is dependent upon the format of the current display. If the current display is in instruction format, pointer format or relative format, the least significant 20 bits of the currently open cell (2 words) is assumed to specify an address. If the current display is in octal, unsigned integer, signed integer, bit, or hex format, the contents of the currently open cell (1 word) is assumed to specify an address. Other formats will be errored.

Usage Notes:

1. When in the modify-active mode, the sub-command NEXT, PREV, OPEN, and any of their abbreviations may be classified by program symbols with the same spelling. When in doubt about a possible conflict, the user can use the activation character or leave the modify-active mode (return key) before issuing any of these sub-commands.

2. All of the MODIFY sub-commands may be issued as major commands in which case their action is based on the last MODIFY command issued.

3. New values for the open cell can be expressed as octal or decimal numbers, simple expressions, address expressions, or instructions in mnemonic form. When the new value is specified followed by a carriage return, the value is stored in the currently open cell, and DELTA leaves the modify-active mode. For example:

```
>MODIFY CPA+.265\I
CPA+.265 / LDPO 3,,PR2 M>TRA CPA+.324
>MODIFY $DS1->.101
$LS4->.101 / .236 M>$L+.40000
```

   If, however, the specifications of the new value is followed by one of the characters N, <LF>, P, |, ↑, O, *, or <TAB>, then the new value is stored in the currently open cell and action is taken according to the descriptions in Table 6-1.

4. These characters may optionally be preceded by a format specification indicating the desired format to be used for the requested display. These characters, with the optional format specification, may be entered either following or in lieu of the new value for the currently open cell. When specified in lieu of any new value, the currently open cell is not modified. For example:

Command	Explanation
>MODIFY .1023 .1023 / .501 M>$L+.4000 [	Add .4000 to current cell and re-display it.
.1023 / .4501 M><LF>	Display the next cell.

```
.1024 / .777777777742 M>\S[       Re-display in signed binary
.1024 / -30 M>                    (decimal).

>MODIFY .2415
.2415 / .5006001 M> <TAB>         Display indirect through
                                  current cell.

.5 / .0 M>.2415\P/                Try it again, this time as
                                  a pointer.

.2415 / .5-0-0,$LS1 M> <TAB>      ...then go indirect.
$LS1->.5 / .120 M>
```

Although the addresses used in the above examples have been simply expressed, the MODIFY command will accept any address specification that can be resolved to a segment and word offset within that segment. When an address is specified by using variable names, all the qualification rules that apply to the DISPLAY or LET command also apply to MODIFY. However, unlike DISPLAY or LET, only the word address portion of the variable information is used. For example, the MODIFY command will accept a PL-6 variable that is defined as an automatic variable in the current auto frame of type BIT(1); but when displayed the whole word containing the BIT(1) variable will be printed and the address will be printed as a word offset in the segment containing the automatic stack for the current process. For example:

```
>DISPLAY BITVAR
BITVAR = '1'B
>MODIFY BITVAR
$LS4->.106 / .4000012623100 M>
```

When specifying addresses in the run unit's procedure (position reference) when using the debug schema information (statement numbers, internal entry names, and statement levels) it is necessary to precede the address specification with a colon. This causes the address to be interpreted exactly as described for the AT and GO commands. For example:

```
>MODIFY 258                          (treated as decimal word
.410 / .104213 M>                    offset in IS)

>MODIFY :258                         (treated as statement 258
FMN$OPNF+.32 :258 / LDP4 0,,PR0 M>   in the current external
                                     procedure)
```

MODIFY Closed Form Command

Format:

M[ODIFY] location[\f] newcontents [(expected old contents)]

Parameters:

location and f   are the same as in the open form of MODIFY.

newcontents   is an expression that can be evaluated or is a GMAP6 instruction.

expected old contents   must be enclosed in parentheses () and the left parenthesis must be preceded by a blank. The value can be any expression that can be evaluated or a GMAP6 instruction.

Description:

The closed or patching form can be issued either directly or can be attached to a stored command. The difference in this form is that both the address and the new value are specified and the modify-active mode is not entered.

Optionally, a display format can be specified and the expected old value of the addressed location can be specified. The format specifier is used to change the default format for the display of the modified cell. The default format is octal unless the location is in the run unit's procedure space or the new value is specified in mnemonic instruction form, in which case both the previous contents and the new contents are displayed in instruction format. When the specified expected value does not match the previous contents, DELTA treats the command as an interactive MODIFY, displaying the addressed location and prompting for a new value.

Example:

>M MYLOC+.510 TRA CPA+.210 MYLOC+.510 / TRA CPA+.210 (LDP0 0,,PR2)	The previous contents are displayed in parentheses.
>M MYLOC+.510 TRA CPA+.210 (LDP0,,PR3) (LDP2 0,,PR3) EXPECTED VALUE NOT FOUND	Bad expected value.
MYLOC+.510 / LDP0 0,,PR2 M>	DELTA prompts for a new value.
>M MYLOC+.510\O TRA CPA+.510	Use of format specifier.

DUMP Command

Format:

```
                                      [ON      ]
DU[MP][A][L] location[,word count][\f] [OVER fid] ['title']
                                      [INTO    ]
```

Parameters:

A   indicates that ASCII translation is desired.

L   indicates nonsuppresion of duplicate lines is desired.

location   may be any form of location or position reference.

word count   defines the number of cells (beginning at location) to be dumped. If word count is omitted, cells beginning from location to the end of the memory segment are dumped. If missing pages are encountered, they are reported and the dump continues at the next page. Word count may be an expression.

f   designates octal 'O' or hexadecimal 'X' format for the dump. f may also be 'C', in which case DUMP becomes the equivalent of DUMPA.

fid   may be any valid CP-6 file identification.

'title'   is any character string to be printed at the top of the dump.

Description:

The DUMP command dumps a range of memory. The user may optionally specify that:

o Each cell be dumped in either octal or hexadecimal format.

o The dump output be directed to a destination other than the one currently assigned for DELTA's output.

o A title line be printed at the top of the dump.

o The width of the dump output be divided in half such that:

  - the left half is printed in the specified format (octal or hexadecimal)

  - the right half contains the interpretation of all printable characters in ASCII.

o Duplicate lines not be suppressed.

Usage Notes:

1. When fid is specified, the run unit must have a reserved DCB slot to be used by DELTA for the duration of the DUMP command. Absence of the fid option specifies that DELTA will output the dump through the normal output DCB M$DO. ON will cause an error if the file referenced by fid already exists, OVER will overwrite an existing file, INTO will extend an existing file or create a new one if none exists.

2. If fid is LP@WSN, the CP-6 system will release the symbiont file to be printed as soon as the dump is complete; a PRINT command to IBEX is then not needed.

3. The form DUMP location\C is equivalent to DUMPA location. With both forms the dump is octal with ASCII translation.

4. To stop a lengthy DUMP printout, depress the break key.

5. If location is preceded by a ":", then a substatement and offset must be specified before specifying a word count. For example:

```
DUMP :100,1,.0, word count
DUMP :100,,.0,word count
```

FIND Command

Format:

```
F[IND][N[OT]] pattern1 [F[ROM] range] [U[NDER] mask1]
    [S[UB] pattern2] [U[NDER] mask2]
```

Parameters:

NOT specifies a search for cells whose contents are NOT equal to pattern1.

pattern1,pattern2 may be specified in any form that can be resolved by DELTA into a 36-bit pattern. This includes:

o Octal or decimal literals. These are converted to their binary equivalents (right-justified with zero-fill).

o Bit string literals. These are left-justified with zero-fill when necessary.

o Character string literals. These are truncated to four characters, blank-filled on the right if necessary.

o Octal string literal. These are left-justified with zero-fill where necessary. Truncation to a 36-bit value is also performed when necessary.

o GMAP6 assembler instruction expression. Single word instructions only.

mask1,mask2 may be any of the above literal types, but may not be a GMAP6 instruction.

range is expressed in the form:

location[,word count]

where location may be any valid position or location reference.

word count defines the number of cells to be searched. If omitted, all cells from location to the end of the memory segment are searched.

Description:

The FIND command searches a specified range of cells for a specified 36-bit pattern, and reports the address of every cell within the range in which the pattern was found. The user may optionally specify a secondary pattern with which DELTA is to replace every occurrence of the pattern being sought. Both the search and the substitution may be conducted under mask. If the search is conducted under mask, the contents of each cell in the range is logically ANDed with the search mask in a work cell to select those bits which are to be considered in the comparison. The search pattern is also masked in the same way. The work cell is then compared with the masked search pattern. When the comparison proves equal, then:

- o If substitution is indicated and a substitution mask is not present, DELTA replaces the contents of the entire cell with the secondary pattern, and reports the location/position.
- o If substitution is indicated and a substitution mask is present, DELTA selects those bits from the secondary pattern which have corresponding bits set in the substitution mask, inserts them into the cell, and then reports the location/position.
- o If substitution is not specified the position/location is reported along with the cell's contents.

DELTA then continues the search until the end of the range is reached.

Usage Notes:

1. To stop a lengthy FIND press the break key. DELTA will print out the address of the next cell to be compared, and stop. There is no way to continue the search without issuing another FIND command.
2. The FIND command is not allowed for FEP programs.

Special Considerations:

Two of the syntax components of the FIND command are associated with certain of DELTA's distinguished names. These assume a default value when DELTA is invoked which can be changed in two different ways:

- o Implicitly specified as a syntax component in the FIND command.
- o When referenced by its distinguished name in a MODIFY or LET command.

SYNTAX COMPONENT	DISTINGUISHED NAME	DEFAULT VALUE
range	$FRANGE	Entire instruction segment (only modifiable syntactically)
mask1	$FMASK	.777777777777 (-1)

The SUB pattern and mask are not remembered by DELTA and must be specified each time you issue the FIND command and desire substitution to take place. For example:

PATTERN	EXPLANATION
FIND .7	FIND an octal literal (binary 7).
FIND 'ABCD'	FIND a character string literal.
FIND TRA :HERE	FIND a GMAP6 assembler instruction which is a transfer to the statement label HERE in the current compile unit.
FIND TSX1 HERB	FIND a GMAP6 assembler instruction which is a transfer and set index to the external entry name HERB.
FIND NOT 0	Find cells which do not contain zero.

RANGE	EXPLANATION
FROM HERE,50	50 words, beginning with the data cell named HERE.
FROM 0	Entire Instruction Segment (IS).
FROM $LS4->0	Entire Automatic Stack ($LS4, $DS1).
FROM $TCB->0,66	Top TCB frame (66 words long).
FROM MYPROG:20,.30	30 octal words from statement 20 in MYPROG.
FROM .50,.20	20 octal words beginning at location .50 in the Instruction Segment ($LS0).

FIND under default mask and report only:

```
FIND -1 FROM TABLE,100
```

FIND under mask and report only:

```
FIND TRA 0 FROM MAINLINE,1000 UNDER .777000
```

The command is to find all TRA instructions within the range.

Mask is: .000000777000

TRA op code is: xxxxxx710xxx

The 0 in TRA is necessary even though it does not enter into the comparison. TRA is a GMAP6 instruction; TRA alone is a symbol name.

FIND under default mask and substitute:

```
FIND TRA :HERE SUB TRA :THERE
```

This command causes DELTA to replace all branches to HERE with branches to THERE within the default range.

FIND under mask and substitute under mask:

```
FIND '4'O FROM WORD_TABLE,50 UNDER .700000000000
SUB '5'O UNDER .700000000000
```

This command causes DELTA to find each word in the first 50 words of WORD_TABLE which contain a 4 in its high-order octal digit and replaces only that digit with a 5. '4'O and '5'O are expressed as octal literals which are left-justified.

STORE Command

Format:

STO[RE] pattern [F[ROM] range] [U[NDER] mask]

Parameters:

pattern may be specified in any form that can be resolved into a 36-bit pattern. This includes:

- o Octal or decimal literals. These are converted to their binary equivalents (right-justified with zero-fill).
- o Bit string literals. These are left-justified with zero-fill when necessary.
- o Character string literals. These may not exceed four characters. They are converted by DELTA into their binary equivalents (left-justified with space-fill where necessary).
- o Octal string literals. These are left-justified with zero-fill where necessary.
- o Any GMAP6 assembler instruction expression (Single word instructions only).

range is as described under the FIND command. The default is the entire instruction segment.

mask may be any of the above literals, but may not be a GMAP6 instruction. If mask is specified, DELTA selects those bits from the pattern which have corresponding bits set in the mask and inserts them into the cell. The default is .777777777777 (-1).

Description:

The STORE command modifies every cell in a specified range of cells. The contents of the cell may be completely replaced or replacement may be on a bit-by-bit basis under control of a mask.

Usage Notes:

1. The range for STORE is remembered in the same location as the range for FIND ($FRANGE).
2. The STORE command is not allowed for FEP programs.

PMD Command

Format:

PMD optionlist

Parameters:

optionlist   is any list of the following keywords:

AU[TO]	Auto segment (DS1)
CO[MMON]	Common segment (DS2)
DA[TA]	Data pages within Instruction Segment
DC[BS]	All DCBs
DS	All data segments
DS1 through DS8	Data segments 1 through 8
D[YNAMIC]D[ATA]	Dynamic data pages within Instruction Segment
IS	Entire Instruction Segment
JI[T]	User JIT
PL[UGH]	Causes same action as the PLUGH command.
PR[OCEDURE]	Procedure pages within Instruction Segment
RO[SEG]	Entire Read-only segment
TC[B]	Task control block (TCB stack)

Description:

The PMD command causes all or selected portions of a user's entire program image to be dumped. Options allow for general selections of what areas to dump.

Usage Notes:

1. The PMD command is not allowed for FEP programs.

Section 7

Mode Control Commands

The commands discussed under this heading are those used to instruct DELTA to change from the normal debug mode to one of two other modes: RUM and ANLZ. RUM mode is used for applying permanent patches to a run unit file. ANLZ mode is used for examining the running monitor or a system dump file and requires special system privileges.

RUM Command

Format:

```
            [  {NUTS     }] [  {C[OPY] }]
RU[M] fid [, {UTS=value}] [, {I[NPUT]}] [B[UFFERS] = n]
                            [  {T[EST] }]
```

Parameters:

fid   is the file identification of the run unit file.

NUTS   overrides the requirement for creation UTS (Universal Time Stamps) to match. If neither NUTS nor UTS=value is specified, NUTS is assumed. If the run unit is in the system account :SYS, IN must be specified.

UTS=value   specifies an octal digit string of the form mm/dd/yy hh:mm:ss.ss which must match the creation UTS as given to the run unit by the linker. If neither NUTS nor UTS=value is specified, NUTS is assumed. If the run unit is in the system account :SYS, IN must be specified.

INPUT   specifies to DELTA that the run unit file is to be opened in the input mode; therefore, the run unit may only be examined. (If a file is busy or the user does not have write permission to it, the file is opened in INPUT mode.) If neither NUTS nor UTS=value is specified, NUTS is assumed. If the run unit is in the system account :SYS, IN must be specified.

COPY   creates a new copy of the run unit file with the same name. This is handy if the file is already open and the user wishes to modify the run unit without affecting the user of the current generation of the run unit.

TEST   specifies that patches made are not to be permanently applied to the run unit. The run unit will be opened in the input mode.

BUFFERS=n   specifies the number of buffers for DELTA to use. This speeds up I/O. Maximum is 10. Default is 5.

Description:

RUM command causes DELTA to access a run unit file instead of memory for accesses made by the MODIFY, FIND, STORE, and DUMP commands. In this manner the programmer can examine and change (apply permanent patches to) the run unit. Patches made with the MODIFY, FIND, and STORE commands are recorded in the run unit file and may be displayed at any time with the LIST command. The DISPLAY and LET commands cannot be used in this mode.

While in RUM mode the patch symbols @, @@, #, and ## can be used (if patch space has been included in the run unit) and addresses for these symbols will be permanently updated in the run unit for future use. Patch space may be defined in a run unit by including the file B_PATCH from the :LIBRARY account in the object file list or using IPATCH and DPATCH options on the !LINK command when linking the run unit. The special symbols $SA and $PRIV may be modified to change the start address and privilege flags associated with the run unit. The special pointer symbol $HEAD may be used to address any offset within the head record of the run unit.

Usage Notes:

1. When EOM mode is in effect the = and / will activate. Substitution of blanks for the = and / in the UTS option is acceptable to DELTA.

ANLZ Command

Format:

```
       [@                      ]
AN[LZ] [RUNNING [,schemafid]]
       [dumpfid                ]
```

Parameters:

@   specifies the running program.

RUNNING   specifies the running monitor.

dumpfid   is the identification of the dumpfile. If this parameter is omitted, the running monitor is used.

schemafid   is the identification of the schema file. If omitted, M:MON.:SYS is assumed.

Description:

The ANLZ command causes DELTA to access the running monitor or a dump or schema file instead of memory. The command selects the dump file that is to be processed and a file from which to take the schema.

Section 8

Miscellaneous Commands

The following commands, which do not fall into any previous classification are described in this section:

UNFID
XEQ
LIST
HELP
END/QUIT

UNFID Command

Format:

```
          {dcb#    }
UNF[ID]   {dcbname }
```

Parameters:

dcb# is the number of the DCB as assigned by the linker specified as a decimal or an octal value. (Examples: 10 and .12)

dcbname is a DCB name such as M$SI.

Description:

The UNFID command performs the M$UNFID monitor service on the specified DCB and prints the results.

Example:

```
>UNF M$UC
DCB #3 M$UC UC# (OPEN, UPDATE, TERMINAL)
```

XEQ Command

Format:

X[EQ] instruction

Parameters:

instruction must be specified as a GMAP6 instruction mnemonic. X0 through X7 may be used in the tag field to denote indexing. All tag mnemonics are recognized, and in addition the tag field may be specified as a decimal number (0-7 only) to indicate indexing with X0-X7, or the tag field may be given in octal in which case it will be used as given. The PR field may be specified mnemonically with PR0 through PR7 or with the decimal numbers 0 through 7.

Description:

The XEQ command causes the immediate execution of a single word machine instruction. XEQ is not allowed on certain types of instructions such as transfers.

Usage Notes:

1. The XEQ command is not allowed for FEP programs.

LIST Command

Format:

```
LI[ST]
```

Description:

The LIST command will list all changes made to a run unit in RUM mode. The LIST command may be issued while DELTA is in either RUM mode or debug mode.

HELP Command

Format:

```
{H|HELP} [(processor)][ TOPICS ][ keyword1 ][ - ][ keyword2]
```

Parameters:

processor   specifies a program or processor that has an associated helpfile in the form HELP:processor:[.[account][.password]]. The default is the processor currently under control.

TOPICS   specifies that only a list of available topics (message names) is required. The range of topics is determined by keyword1 or keyword2.

keyword1   specifies the name of the message to be printed.

keyword2   if a range is specified (by including a dash "-"), keyword2 is the upper limit of the range. If a range is not specified, keyword2 is a submessage, or category within a message.

Description:

HELP prints information.

HELP messages have levels. Once the initial level has been printed, typing a question mark prints the next level, usually containing greater detail. Typing two question marks prints the entire message.

Any standard HELP file (even those for other processors) can be accessed with the HELP command.

Example:

```
    HELP (DELTA) RUM
```

Usage Notes:

1. The standard CP-6 HELP facility is used by DELTA so the user can get help on any other CP-6 processor by entering:

```
    HELP (processor) [TOPICS] [keyword1] [-] [keyword2]
```

END/QUIT Commands

Format:

```
{EN[D]  } [fprg-res]
{Q[UIT]}
```

FEP Parameters:

fprg-res	will cause the specified FEP program to be terminated.

Description:

The commands EN[D] and Q[UIT] cause the current DELTA session to be terminated, and returns the user to his command processor (IBEX).

Section 9

Debugging FEP Programs

This section discusses debugging FEP programs with DELTA. Included are the following:

- o New commands pertaining just to debugging FEP programs - (DEBUG, USE, WAIT, STOP).
- o Changes in defaults when debugging FEP programs.
- o Changes to existing commands - FEP program debugging specific options.
- o Special Distinguished Names for debugging FEP programs.

Debugging Forms Programs

When debugging a forms program, two terminals are required. One terminal is logged on as a station that runs the FPL program; the other is a timesharing terminal that runs DELTA.

DELTA may be invoked prior to or during Forms Program execution. All debugging sessions begin by starting the Forms at a TP station. Then DELTA is invoked from a separate timesharing terminal. As soon as DELTA is activated, it takes control of Forms Program execution.

Associating DELTA at Program Invocation

Associating DELTA at program invocation requires entry of special commands at the TP station and the timesharing terminal.

TP Station Debug Command

Format:

DEBUG name

Parameters:

name	is the Forms Program name (as entered in the BEGIN command).

Description:

The DEBUG command—instead of the BEGIN command—is entered at the TP station to invoke the Forms Program for debugging. As a result of this command, the Forms Program is loaded, but does not begin execution.

At a separate timesharing terminal, the user invokes DELTA and DELTA responds with a prompt. The user then enters the DEBUG command to associate DELTA with the station and comgroup that invoked the Forms Program for debugging.

DEBUG Command

Format:

```
DEB[UG] station {AT} {comgroupfid}
                {ON} {TP/tpacct   }
```

Parameters:

station   is a character string of eight characters or less that defines the station name of the terminal running the Forms Program to be debugged. If station is all numeric, it must be entered in quote strings.

comgroupfid   is a comgroup fid of the form CG[#psn]/name.acct or of the form CG[#psn] name.acct. 'name' identifies the comgroup through which the FPL program is communicating.

tpacct   is a character string of eight characters or less that names the account in which the particular instance of TP is being run.

Description:

This command associates DELTA with the TP station. The user can then enter DELTA commands, including GO to start execution of the Forms Program.

Example:

At STATION1 on the instance of TP called PAYROLL, the user enters:

```
    DEBUG PAY
```

At the timesharing terminal the user enters:

```
    !DELTA
    DELTA C00 HERE - NO RU ASSOCIATED
    >DEBUG STATION1 AT TP/PAYROLL
```

In this example, the Forms Program called PAY is loaded and halts. Then DELTA is invoked to debug the Forms Program operating on behalf of STATION1 on the PAYROLL instance of TP (i.e., PAY).

Post-associating DELTA

After a Forms Program has been invoked via the BEGIN command, the user can associate DELTA. At a timesharing terminal, the user invokes DELTA and enters the DEBUG command as shown above. DELTA assumes control over the execution of the Forms Program, the user may then enter DELTA commands.

Usage Notes:

1. If a second DEBUG command is entered before the user QUITS or ENDS the first program being debugged, DELTA will do a KILL DELTA on the first program and then set up for debugging the second specified program.

Debugging DCB-Connected FEP Programs

DCB-connected FEP programs are started via an M$SETFP monitor service call done in a program running on the Host. In order to debug DCB-connected FEP programs, the user just starts the Host program under DELTA.

```
!U
!HOSTRU.
```

or

```
!START HOSTRU UNDER DELTA
```

This associates DELTA with the Host program and with any FEP programs started via an M$SETFP monitor service call by the Host program. DELTA will identify which program the user is dealing with by preceding the prompt ('>') with the value the user specified for RES on the M$OPEN of the FEP program.

```
Example                                        Explanation

 !U
 !HOSTRU.
 DELTA C00 HERE IC=TEST:1 [PROC]
 >G                                            User does GO to host
                                               program so M$SETFP can
                                               happen.

 HOST PROGRAM STOPPED IC=TEST:4956 [MONITOR CALL]
 (UC05) IC=A:1 [PROC]
 FPGR STARTED VIA M$SETFP
 UC05>                                         User gets control when
                                               FEP program (UC05) gets
                                               started - user can enter
                                               DELTA commands for this
                                               FEP program, i.e., set
                                               breakpoints, GO, etc.
```

Post Association of DELTA

To associate DELTA with a Host program and the FEP programs started by the Host program via an M$SETFP monitor service call, the user types CTRL Y. IBEX prompts with a double bang (!!). The user types DELTA to associate DELTA with the interupted Host program and the FEP programs associated with the Host program. Any FEP programs which have been started by the Host program will be reported to the user. When DELTA is associated, if the host program or any FEP program is autoshared, the user should use the UNSHARE command if any modification of procedure (including breakpoints) is to be done.

```
Examples                      Explanation

 !HOSTRU.                     Program does not prompt as expected;
                              a loop is suspected. The user enters
                              <CTRL><Y>.

 !!DELTA
 DELTA C00 HERE IC=TEST:5053,,.3 [ASSIGNMENT]
 (UC05) IC=A:10 [ASSIGNMENT]
 >
```

Debugging a Specified SYSID on a Specified FEP

DELTA may be associated with a FEP program (FPRG) by specifying the sysid and fepname of the program on the DEBUG command.

DEBUG Command

Format:

```
DEB[UG] sysid {AT} fepname
              {ON}
```

Parameters:

sysid   is the decimal sysid of the desired FEP program to debug.

fepname   is a character string of eight characters or less that identifies the FEP in which the program to debug is running.

Description:

This command associates DELTA with the FEP program identified by its sysid and the FEP in which it is running. The user can then enter DELTA commands, including GO to start execution of the FEP program.

Example:

At a timesharing terminal the user enters

```
    !DELTA
    DELTA C00 HERE - NO RU ASSOCIATED
    >DEBUG 21 AT FEP12
```

DELTA is associated with the FEP program in FEP12 that is sysid 21.

Usage Notes:

1. If a second DEBUG command is entered before the user QUITs or ENDs the first program being debugged, DELTA will do a KILL DELTA on the first program before setting up for debugging the second specified program.

WAIT Command

Format:

```
WA[IT]
```

Description:

The WAIT command causes DELTA to wait for the next event from an FEP program—i.e., hitting a breakpoint, trap, etc. This command is especially useful after the user has done a GO to one or two FEP programs, does not want the Host program to continue execution, and wants to wait for the next breakpoint or trap in one of the executing FEP programs.

STOP Command

Format:

```
STOP {A[LL]    }
     {fprg-res}
```

Parameters:

ALL   specifies all FEP programs.

fprg-res   specifies a particular FEP program. This is the value the user specified for RES on the M$OPEN of the FEP program.

Description:

The STOP command will cause the specified FEP program(s) to be interrupted. DELTA will report the location at which any FEP programs were interrupted to the user and prompt for commands.

Example:

```
>STOP ALL
(UC05) IC=A:10,,.3 [ASSIGNMENT]
(UC10) IC=B:310,,.4 [CALL]
>
```

User can now enter DELTA commands.

USE Command

Format:

```
U[SE] {fprg-res}
      {H[OST]   }
```

Parameters:

fprg-res   specifies a particular FEP program. This is the value the user specified for RES on the M$OPEN of the FEP program.

HOST   specifies the host program.

Description:

The USE command is used to specify a particular program the user wants to examine. If the specified FEP program is running, it will be stopped for the user to examine it.

Example:	Explanation:
>USE UC05	This specifies the FEP program identified by UC05.
>USE HOST	This specifies the host program.

Usage Notes:

1. When a USE command is done by the user, DELTA will remember the state of the program being USEd and if another USE command is done without GOing the first program, the first program will be returned to its original state.

Defaults when Debugging FEP Programs

- o A number preceded by a period is a hexadecimal number when debugging FEP programs (it is an octal number when debugging Host programs).
- o A maximum of 20 breakpoints are allowed for a single FEP program.

Existing Command Changes

Several of the existing DELTA commands have new options which pertain to debugging FEP programs. Below is a list of the commands and a summary of the new options. For more detail, see the desired command.

Command	New Options
GO	fprg-res – GO a specific FEP program ALL – GO ALL FEP programs HOST – GO the host program
QUIT	fprg-res – QUIT a specific FEP program
KEEP/TRAP/IGNORE	fprg-res – identifies a specific FEP program ALL – identifies ALL FEP programs HOST – identifies the Host program
KILL	DELTA fprg-res – disassociates DELTA from the specified FEP program
SHOW	FPRG's – shows state of all FPRGs – if stopped, shows current IC fprg-res – shows state of a specific FEP program
MODIFY	M[ODIFY] D[OUBLE] [W[ORD] – Modify two words – specifically tells DELTA you want to modify two words.

Distinguished Names

DELTA recognizes many distinguished names related to FPRGs as syntactical components of DELTA's commands. These names are listed below.

Included are the machine registers visible to programming and usage and default formats for display. D, L, M, and P indicate DISPLAY, LET, MODIFY, and POINTER usage respectively. When one of these is used in a pointer qualified reference, it must be the leftmost name in the expression.

NAME	REFERS TO:	USAGE	DEFAULT FORMAT
$B1 to $B7	Corresponding 16-bit Base Register	P,D,L,M	Hex
$R1 to $R7	Corresponding 16-bit General Register	D,L,M	Hex
$M1 to $M7	Corresponding 8-bit Mode Control Register	D,L,M	Hex

$T	Stack address Register (16 bits)	D,L,M	Hex
$S	System Status Register (16 bits)	D,L,M	Hex
$	Address associated with a trap (20 bits)	D,L,M	Hex
$IV	Contents of $B3 at the time of a trap (16 bits)	D,L,M	Hex
$Z	Z-word-miscellaneous information relative to a trap (16 bits)	D	Hex
$ASV	Address Space Vector - the base of an MPU image in memory (20 bits)	D	Hex
$CI	CIP Indicator Register (8 bits)	D,L,M	Hex
$IND	Indicator Register (8 bits)	D,L,M	Hex
$IC or $P	Instruction Counter (20 bits)	D,L,M	Relative
$ISR1 to $ISR7	Instruction Segment	P,D	Descriptor
$AUTO	Auto Segment	P,D	Pointer
$DS1 to $DS4	Corresponding LCP6 Data Segment descriptor	P,D	Descriptor

System Areas

This group of distinguished names are used exclusively as pointers to various system areas.

NAME	REFERS TO:
$ROSEG	Read-Only-Segment
$RDBR	Remote descriptor array
$TSA	Trap Save Area
$DCBn	Refers to DCB #n

Appendix A
Distinguished Names

DELTA recognizes many distinguished names as syntactical components of DELTA's commands. These names are listed below.

Included are the machine registers visible to programming, and usage and default formats for display. D, L, and M indicate DISPLAY, LET, and MODIFY, respectively.

NAME	REFERS TO:	USAGE	DEFAULT FORMAT
$A	Bits 0 to 35 of Accumulator (AQ)	D,L,M	Octal
$AU	Bits 0 to 17 of Accumulator	D,L,M	Octal
$AL	Bits 18 to 35 of Accumulator	D,L,M	Octal
$Q	Bits 36 to 71 of Accumulator	D,L,M	Octal
$QU	Bits 36 to 53 of Accumulator	D,L,M	Octal
$QL	Bits 54 to 71 of Accumulator	D,L,M	Octal
$E	Exponent Register (8 bits)	D,L,M	Octal
$EA	Exponent and A Registers	D	Floating Point Single Precision
$EAQ	Exponent, A, and Q Registers	D	Floating Point Double Precision
$IND	Indicator Register (18 bits)	D,L,M	Octal
$X0 to $X7	Corresponding 18-bit Index Registers	D,L,M	Relative
$IC	Instruction Counter (18 Bits)	D,L,M	Relative

Program Visible Descriptors

These distinguished names are used to refer to descriptors either in machine registers or in defined positions in segments. P, D, and M indicate POINTER, DISPLAY, and MODIFY usage, respectively. When one of these names is used in a pointer qualified reference, it must be the leftmost name in the expression.

NAME	REFERS TO:	USAGE	DEFAULT FORMAT
$SSR	Safe Store Register (When used as a pointer $SSR points to the user's safe store frame as placed on DELTA's TCB.	P,D	Descriptor
$ISR	Instruction Segment Register	P,D	Descriptor
$ASR	Argument Segment Register	D	Descriptor

$LSR	Linkage Segment Register	D	Descriptor
$PSR	Parameter Segment Register	D	Descriptor
$LS0 to $LSn	Descriptor n in Linkage Segment	P,D	Descriptor
$AS0 to $ASn	Descriptor n in Argument Segment	P,D	Descriptor
$PS0 to $PSn	Descriptor n in Parameter Segment	P,D	Descriptor
$DS1 to $DS8	Corresponding CP-6 Data Segment Descriptor (DS1 = LS4, etc.)	P,D	Descriptor
$DR0 to $DR7	Corresponding Operand Descriptor Register	P,D	Descriptor
$PR0 to $PR7	Corresponding Pointer Registers (DRn, SEGIDn, ARn)	P,D,M	Descriptor
$AR0 to $AR7	Corresponding Address Registers	D,M	Addr Register (AR)

XDELTA and ANLZ Only

With the exception of $SSR, the distinguished names in this group are addressable only by XDELTA and the ANLZ mode of DELTA.

NAME	USAGE
$WSR0 to $WSR7	Refers to the corresponding Working Space Register. May be used in DISPLAY, MODIFY, and LET commands. The default display format is in decimal.
$WSQ0 to $WSQn	Refers to word n of the Working Space Page Table Directory (WSPTD). May be used in the DISPLAY command.
$SSR	Refers to the Safe Store Register for the current domain (see USE command). May be used as a pointer or may be displayed (descriptor).
$REAL	May only be used as a pointer to address real memory.
$VIRT	May only be used as a pointer to address virtual memory.
$RVB	Refers to the Recovery Buffer. May only be used as a pointer (ANLZ file mode only).
$USD	Refers to the User Directory. (ANLZ file mode only). May only be used as a pointer.
$HJIT	Refers to the Housekeeping Job Information Table for the current domain. May only be used as a pointer.
$CGn	Refers to nth Communications Group area of system dump file. May only be used as a pointer. (ANLZ file mode only).

RUM Mode Only

The distinguished names in this group are addressable by the RUM mode of DELTA.

NAME	USAGE
$SA	Refers to start address in head record.
$PRIV	Refers to privilege bits in head record.
$HEAD	May be used as a pointer to any offset in the head record.

System Areas

This group of distinguished names are used exclusively as pointers to various system areas.

NAME	REFERS TO:
$TCB	Top Frame of the Task Control Block
$JIT	Job Information Table
$PARMn	Parameter #n of PL-6 External Procedure ($PR2->n+2)
$DCBn	Refers to DCB #n

Command Work Areas

This group of distinguished names refers to various work areas used by certain commands.

NAME	USAGE
$I	Refers to the position indicated by the current Instruction Counter. $I may be used in a MODIFY Expression (e.g., MODIFY $I+3).
$O	Refers to the last word of memory that was modified. Only the MODIFY command (along with N, P, and *) changes the value of $O.
$L	Refers to the last value displayed by DELTA as a result of a MODIFY type command.
$FRANGE	Refers to the range specification of the FIND/STORE displayed. It may only be displayed.
$FMASK	Refers to the Find Mask (mask1) in the FIND command.

Patching Symbols

This group of distinguished names refers to various work areas which are managed by DELTA when the RUM Mode is invoked.

NAME	USAGE
@	Refers to the next word in the procedure patch space.
@@	Refers to the next even word in the procedure patch space.
#	Refers to the next word in the data patch space.
##	Refers to the next even word in the data patch space.
$RI	Replaced Instruction. Refers to the contents of the last modified memory word not contained within either the data or procedure patch space.
$NA	Next Address. Refers to the address of the word of memory immediately following $RI.
$REM@	Used to DISPLAY the remaining amount of Procedure patch space.
$REM#	Used to DISPLAY the remaining amount of Data patch space.

These symbols may also be used in the debug mode to make temporary patches to the run unit image in memory.

Appendix B

Assembler Instruction Formats for Patching

This set of Assembler Instruction Formats is used for patching in RUM or DEBUG modes.

DELTA handles the standard GMAP6 instruction format for 1-word instructions except for the following:

- o The ZERO op code has the following format:

  ```
  ZERO    any DELTA position or location reference, octal constant.
  ```

  The ZERO op code specifies the contents of a word of memory. The evaluated position or location reference is placed in the left half of the word and the octal constant is placed in the right half. If the position or location reference is preceded by a ":", a substatement and offset must be specified before specifying the octal constant. For example:

  ```
  ZERO :100,1,.1,.1000
  ZERO :100,,,.1000
  ```

- o The PTR op code has the following format:

  ```
  PTR    any DELTA position or location reference, octal constant.
  ```

  The PTR op code specifies the contents of a word of memory in pointer format. The only segids which may be specified are .6000-.60FF. The octal constant specifies 00-FF of the segid. The evaluated position or location reference is placed in the left half of the word. It is not possible to specify a character or bit offset.

- o A "!" after the op code causes DELTA to set the inhibit bit in the generated instruction word. For example:

  ```
  LDA! .4010,DL
  ```

Multiword instructions must be specified to DELTA in octal. No mnemonic form is available for the instruction word or the descriptors. When DELTA prints a multiword instruction, the octal value of the instruction word is preceded by the op code mnemonic.

The following multiword instruction:

```
MLR FILL='040'0
ADSC9  2,QL,PR1
ADSC9  TEXT
```

must be entered to DELTA as:

```
.40000100506
.100002000010
.1000010
```

and will be printed out by DELTA as:

```
MLR  .40000100506
 .10000200010
 .1000010
```

Note: The descriptor words may be printed in octal or some single instruction format. The user must specify \O format to force printing of the descriptor in octal.

For code that is the subject of XEC or XED, or that is generated dynamically, it is advantageous to keep DELTA away from it, i.e., do not use AT, STEP, XEQ, TRACE TRANSFER in such code, unless the user knows how DELTA will modify the code to do these commands.

Index

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE	CP-6 DELTA REFERENCE MANUAL	ORDER NO.	CE39-03
		DATED	MARCH 1985

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME ______________________ DATE ____________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

____

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

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

Together, we can find the answers.

Honeywell

Honeywell Information Systems
U.S.A.: 200 Smith St., MS 486, Waltham, MA 02154
Canada: 155 Gordon Baker Rd., Willowdale, ON M2H 3N7
U.K.: Great West Rd., Brentford, Middlesex TW8 9DH **Italy**: 32 Via Pirelli, 20124 Milano
Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F. **Japan**: 2-2 Kanda Jimbo-cho, Chiyoda-ku, Tokyo
Australia: 124 Walker St., North Sydney, N.S.W. 2060 **S.E. Asia**: Mandarin Plaza, Tsimshatsui East, H.K.

41934, 5C185, Printed in U.S.A.

CE39-03