Preface ¦

This document is intended to be ultimately expanded into a ¦
User's Guide to DAD. In the mean time, it is primarily a ¦
command summary with some explanation of needed concepts in an ¦
attempt to provide enough information so that people can use ¦
the debugger. ¦

Statements that are marked with a vertical bar character (¦) ¦
in the right margin have been editted since the last release ¦
(10 Sep 76 3rd Draft) of this document. Most of these ¦
changes are internal cosmetic changes. The only content ¦
changes are the addition of the two appendices and the ¦
changing of the table of contents to refer to page numbers ¦
rather than statement numbers. ¦

Table of Contents page

Syntax Conventions Used In This Document ¦ 1

With the exception of the formal definition of an address list (which uses a modified BNF), the following syntactical conventions are adhered to in the command summaries:

Command words appear as first letter of the word being upper case and the rest of the word in lower case. When a generic function character (discussed below) is a command word, it will be surrounded by double quotation marks ("...").

Noise words appear as lower case words enclosed in parentheses.

Alternative paths through a rule or command appear as statements at the same level in a plex.

The end of a command or rule is indicated by a colon (:).

An upper case word preceded by an atsign (@) is a reference to a rule described elsewhere.

An upper case word preceded by an uparrow (^) is a reference to a selection entity. Selection entity types (text, character, etc.) are listed in a separate branch.

An uppercase word preceded by an asterick (*) refers to that character currently serving the generic function (discussed below) specified.

An uppercase word not preceded by an atsign or uparrow is a builtin CLI construct (e.g. OK).

Angle brackets (<>) are used to inclose single character keystrokes (e.g. <LINEFEED> refers to hitting the linefeed key on a terminal).

Concepts 2

Entering and Leaving DAD 2a

The following discussion is relevalant to the current release of the debugger and may change in the future.

To use DAD, run the TENEX subsystem <ARCSUBSYS>DAD. When DAD starts, it will do some initialization and then prompt you with the DAD hereald followed by the prompt for a command. DAD's command language is context dependent, and until you have specified a tool for DAD to debug, only a few global commands will be available. Probably the most useful command at this time is the Debug command in which you specify which tool (i.e. which TENEX subsystem) you wish to debug. After specifying a tool, the full compliment of DAD's commands will be avaiable. At this time you may set breakpoints, examine memory, etc. When you are ready to start execution, give the Continue command and execution will start at the tool's main entry vector location. (If you do not wish to start at the main entry vector location, you may use some of the sub-commands of the Continue command.)

To get back to DAD later (in case you forgot to set any breakpoints, or your program is looping, etc.), use the ^L facility. Control-L is a deferred pseudo-interrupt (PSI), which means that you won't actually enter the debugger until the control-L is read. If you wish to enter the debugger immediately, type 2 control-Ls without any intervening typein. To continue execution of what was happening before you re-entered DAD use the Continue command.

When you are through debugging, you may either enter a ^C or use DAD's Quit command. If you are through debugging a specific (instance of a) tool and wish to debug (a different instance of or) another tool, use the Done command which will ask you for a new tool to be debugged after cleaning up the previous tool.

Tools, Processes, Languages, Etc. 2b

The debugger is designed to be a multi-tool, multi-process, multi-machine, multi-high level language debugger. What this glorious statement means is:

The debugger is able to debug more than one tool at a time, and in fact, able to debug tool interactions.

(If you don't know what a tool is, you are in trouble, but see other NSW documentation for a definition.)

Any tool is allowed to contain any internal process structure it desires, and the debugger is able to debug more than one process.

(Process is being used in the conventional computer science meaning of the word.)

Some tools may be capable of running on more than one machine, or different tools may run on different machines. The debugger is capable of coping with this situation.

And lastly, the debugger is capable of accepting from, and presenting to, the user data structures in a format that resembles the (hopefully) high level language in which the tool was coded.

To cope with this multitude of entities, the debugger uses the concept of an Internal Debugger Handle (IDH). An IDH is an unique (per debugging session) positive integer. Each process that the debugger knows about is assigned an IDH. A user may always refer to a process by its IDH, and, in some commands, if the process is the top process for a tool, the user may also refer to it by the tool's usename (see NSW documentation).

A process is assigned an IDH when the debugger first learns of the process. When the debugger is first pointed at a tool, it will determine the process structure for that tool and assign an IDH for each process. Thereafter, the debugger will monitor the tool's execution, and will assign new IDHs to newly created processes at the time they are created.

The debugger is not capable of debugging a tool that is split on more than one machine. This means that the debugger is capable of debugging a tool on one machine for one instance of the tool, and on a different machine for a separate instance of the tool, but for any one instance, the entire tool must exist on one, and only one, machine.

At any instance, the debugger can be pointed at one, and only one, process. This process will be referred to as the current or active target process. This does not mean that the debugger can not know about more than one process, nor that the debugger is not capable of varying the current target process over time. It just means that at any instance, all commands are refering to the current process (with the obvious exception of the Debug command to point at another process). During a debugging session, when a breakpoint is encountered, the process containing the breakpoint will automatically be made the current target process, regardless of which process was current previously.

Character Sets and Generic Functions ¦ 2c

Since the debugger is designed to support a number of different languages, and since most languages do not use the same character sets as valid characters in identifiers, etc., it is not possible for the debugger to always use the same character to mean the same thing in a command. For example, a semi-colon character may be a valid character in an identifier in some languages, and it cannot therefore be used to separate address ranges (discussed below, but for now they are composed of expressions which may contain identifiers) in an address list. Therefore the debugger has adopted the concept of a generic fuction and a generic function character (GFC). A GFC is that character which is currently serving a specific generic function.

For documentation and communication purposes, it is convienient to have a generic name to refer to the character that is currently serving a specific generic function. Thus, while the specific character may change, it can still be refered to by its generic name. The generic name for a character is the uppercase word of the generic function symbolic name preceded by an asterick, e.g. the generic name for the GFC that is currently serving the generic function of an address list delimiter (semicolonchar) is *SEMICOLONCHAR.

The current values of each GFC can be determined by using the Character (set) Display command.

The symbolic names and the meaning of these generic functions are as follows (the debugger default character, in the absence of user or Language Module modification, for a generic function will appear under the meaning column delimited by a left angle bracket (<) and a right angle bracket followed by a semicolon (>;):

generic function symbolic name	meaning of character
pluschar	<+>; the user is using this character as the arithmetic addition operator
minuschar	<->; the user is using this character as the arithmetic subtraction operator

timeschar	<*>; the user is using this character as the arithmetic multiplication operator
dividechar	<'>; the user is using this character as the arithmetic division operator
lparenchar	<(>; the user is using this character as the arithmetic left grouping character
rparenchar	<)>; the user is using this character as the arithmetic right grouping character
blockchar	<&>; the user is using this character as a block delimiter; e.g. the string: string1&string2 should be interpretted as symbol string2 in block string1 if & is the current BLOCKCHAR
fieldchar	<.>; the user is using this character to delimit the fields of a record
escapechar	<ALTMODE or ESCAPE>; the user is using this character to mean interpret the next character as a debugger builtin variable; e.g., ESCAPECHAR followed by a 'Q (or 'q) refers to the builtin debugger variable which has the value of the last displayed cell
lmchar	<%>; the user is using this character to mean interpret the next character(s) as a language module builtin variable or construct
commachar	<,>; the user is using this character as an address range delimiter to separate the two elements of an address range; under

	normal circumstances, a LM will never see the COMMACHAR in user input strings
semicolonchar	<;>; the user is using this character to separate address ranges within address lists
larrowchar	<_>; the user is using this character as the debugger assignment character
tabchar	<tab>; the user is using this character to mean display the cell addressed by the most recently displayed cell
poundchar	<#>; the user is using this character to mean back up to the previous displayed cell
lfchar	<LINEFEED>; the user is using this character to mean display the next sequential cell
uparrowchar	<^>; the user is using this character to mean display the previous sequential cell
bslashchar	<\>; the user is using this character to mean display an address list in string mode
equalchar	<=>; the user is using this character to mean display the value of the input address list
excmarkchar	<!>; the user is using this character to mean display cells as ascii values
lsquarechar	<[>; the user is using this character to mean display an address list numerically
qmarkchar	<?>; the user is using this character to mean tell where symbols in an address

list	are defined
rsquarechar	<]>; the user is using this character to mean display an address list as records
slashchar	</>; the user is using this character to mean display an address list symbollically

User Input and Debugger Output ¦ 2d

All communication with the debugger is governed by the values of 4 records: the permanent and current input mode records, and the permanent and current output mode records. At the beginning of most commands (exceptions discussed below) the permanent input and output mode records are copied to the current input and output mode records, and thereafter the command is governed by the value of these current records.

For example, all numbers entered by the user will be interpretted as being numbers in the base specified by the current input mode radix, and all numbers displayed to the user will be formatted to conform to the current output mode radix.

(The only exception for these specific examples is that when specifying or viewing these radixes, the radix will always be interpretted as being decimal numbers.)

The values of the permanent input and output mode records can be displayed and viewed via the Typeout (mode) and Input (mode) commands.

Several commands provide for modifying the current input and/or output mode records for a specific instance of a command. These ephemeral values are then lost at the start of the next command (see exceptions deiscussed below).

The exceptions mentioned above refer to the commands that consist of a single GFC, e.g. the assign command as entered by *LARROWCHAR. These commands will use the current values of the input and output mode commands at their beginning, i.e. the values of these records that were in effect for the previous command.

Address Lists 2e

Discussion 2e1

An address list is the basic manner in which a user refers to elements in the current target process. Basically, an address list is composed of one or more address ranges; and an address range consists of one or two address range elements (AREs).

(The character that terminates an address list, while it may modify the functional use of the address list, is not a part of the address list itself.)

Address List Terminators 2e2

The user may terminate an address list with a number of different characters, depending on which command is being specified. The terminating character is NOT a part of the address list. The following are the generic characters, with their meaning, that may be used to terminate various address lists:

generic character terminator	meaning
*LARROWCHAR	after each line of the address list is displayed, the user wishes to assign a new value to the just displayed entity
*BSLASHCHAR	the user wishes to see the address list displayed in string mode
*EQUALCHAR	the user wishes to have the value of the input address list displayed to him
*EXCMARKCHAR	the user wishes to see the address list displayed in ascii mode
*LSQUARECHAR	the user wishes to see the address list displayed in numeric mode
*QMARKCHAR	the user wishes to find out where the symbols in the entered address list are defined
*RSQUARECHAR	the user wishes to see the address list displayed in record mode
*SLASHCHAR	the user wishes to see the address list displayed in symbolic mode

*TABCHAR	after displaying the current address list, the user wishes to see the cell (or data structure) addressed by the last dipslayed cell
*POUNDCHAR	after displaying the current address list, the user wishes to see the cell (or data structure) taht was displayed immediately prior to the last cell (or data structure) (i.e. will perform an inverse *TABCHAR)
*LFCHAR	after displaying the current address list, the user wishes to see the cell (or data structure) whose address is one greater than the last displayed cell (or data structure)
*UPARROWCHAR	after displaying the current address list, the user wishes to see the cell (or data structure) whose address is one less than the last displayed cell (or data structure)

Formal Definition

2e3

```
ADRLIST := ADRRANGE [ *SEMICOLONCHAR ADRLIST ]  /  NULL
ADRRANGE := RANGE / BUILTIN / RECORDSPEC
BUILTIN :=
   FRAME / LOCAL / PARAM / SIGNAL / CATCH / MEM / PLIST
   / JFN / ERR
ERR := *ESCAPECHAR ('E / 'e)
JFN := AJFN / RJFN
AJFN := *ESCAPECHAR ('J / 'j)
RJFN := AJFN NUMBER [*COMMACHAR AJFN NUMBER]
PLIST := *ESCAPECHAR ('Z / 'z)
MEM := AMEM / RMEM
AMEM := *ESCAPECHAR ('M / 'm)
RMEM := AMEM NUMBER [*COMMACHAR AMEM NUMBER]
CATCH := *ESCAPECHAR ('C / 'c)
SIGNAL := *ESCAPECHAR ('S / 's)
PARAM := *ESCAPECHAR ('P / 'p)
LOCAL := *ESCAPECHAR ('L / 'l)
FRAME := FSPEC [ *COMMACHAR FSPEC ]
FSPEC := FF / FR / FO / FT / FB / FA
FF := *ESCAPECHAR ('F / 'f)
FR := *ESCAPECHAR ('F / 'f) ('+ / '-) [ NUMBER ]
FO := *ESCAPECHAR ('F / 'f) ('O / 'o)
FT := *ESCAPECHAR ('F / 'f) ('T / 't)
FB := *ESCAPECHAR ('F / 'f) ('B / 'b)
FA := *ESCAPECHAR ('F / 'f) '@ NUMBER
RECORDSPEC := EXPRESSION *FIELDCHAR EXPRESSION
RANGE := EXPRESSION [ *COMMACHAR EXPRESSION ]
EXPRESSION :=
   an expression following the syntactical rules of the
   current language module, usualy composed of sums of
   terms, in which terms are composed of IDENTs
IDENT := BLCKIDNT / SMPLIDNT / NUMBER / BLTNTRM /
METAIDNT
BLCKIDNT := SMPLIDNT *BLOCKCHAR SMPLIDNT
SMPLIDNT :=
   a string composed of valid identifier characters for
   the current language
METAIDNT := *LMCHAR SMPLIDNT
BLTNTRM := BA / BB / BLN / BPN / BQ / BR / BY
BA := *ESCAPECHAR ('A / 'a)
BB := *ESCAPECHAR ('B / 'b) NUMBER
BLN := *ESCAPECHAR ('L / 'l) NUMBER
BPN := *ESCAPECHAR ('P / 'p) NUMBER
BQ := *ESCAPECHAR ('Q / 'q)
BR := *ESCAPECHAR ('R / 'r)
BY := *ESCAPECHAR ('Y / 'y)
NUMBER := a string of digits in the current input mode
radix
```

Semantics 2e4

ADRLIST := ADRRANGE [*SEMICOLONCHAR ADRLIST] / NULL

the NULL address list is equivalent to entering the last input address list

ERR := *ESCAPECHAR ('E / 'e)

used to show the last operating system error incurred by the current target process

AJFN := *ESCAPECHAR ('J / 'j)

used to display an indication of the files being used

RJFN := AJFN NUMBER [*COMMACHAR AJFN NUMBER]

used to display an indication of the files being used for file numbers NUMBER [to NUMBER]

PLIST := *ESCAPECHAR ('Z / 'z)

used as a shorthand notation to be equivalent to the previously typed in address list

AMEM := *ESCAPECHAR ('M / 'm)

used to show the utilization of the address space of the target process

RMEM := AMEM NUMBER [*COMMACHAR AMEM NUMBER]

used to show the utilization of the address space of the target process for pages NUMBER [to NUMBER]

CATCH := *ESCAPECHAR ('C / 'c)

used to show the catchphrases for the current frame.

SIGNAL := *ESCAPECHAR ('S / 's)

used to show the signal status of the process.

PARAM := *ESCAPECHAR ('P / 'p)

used to show the formal parameters of the current frame

LOCAL := *ESCAPECHAR ('L / 'l) ¦

used to show the local variables of the current frame

FF := *ESCAPECHAR ('F / 'f) ¦

FF refers to the current frame. the current frame is the most recently displayed frame or the frame on the top of the stack after the debugger is entered

FR := *ESCAPECHAR ('F / 'f) ('+ / '-) [NUMBER] ¦

if NUMBER is not specified it defaults to 1; no spaces may precede NUMBER; NUMBER specifies the number of frames to move relative to the current frame; e.g. if '$ is the current *ESCPAECHAR, and ', is the current *COMMACHAR, the FRAME: "$ft, $f-2" would display the frame on the top of the stack, and the next two frames towards the bottom of the stack in the control thread.

FO := *ESCAPECHAR ('F / 'f) ('O / 'o) ¦

used to show the owner frame of the current frame; the owner of a procedure is its caller; the owner of a coroutine is the routine that did the openport to the coroutine.

FT := *ESCAPECHAR ('F / 'f) ('T / 't) ¦

used to show the top frame on the stack

FB := *ESCAPECHAR ('F / 'f) ('B / 'b) ¦

used to show the bottom frame on the stack

FA := *ESCAPECHAR ('F / 'f) '@ NUMBER ¦

used to show the frame whose mark is NUMBER

RECORDSPEC := EXPRESSION *FIELDCHAR EXPRESSION |

RECORDSPEC is used to represent a field (specified by the second EXPRESSION) of the record instance at the address specified by the first EXPRESSION; e.g. if period is the current *FIELDCHAR, then the RECORDSPEC: "rec.fld" refers to field "fld" of the record instance at address "rcd".

BLCKIDNT := SMPLIDNT *BLOCKCHAR SMPLIDNT |

BLCKIDNT is used to refer to the (local) symbol (specified by the second SMPLIDNT) in the block (or file) specified by the first SMPLIDNT; e.g. if '& is the current *BLOCKCHAR, then the BLCKIDNT: "fl&sfilev" would refer to the symbol "sfilev" in file "fl".

METAIDNT := *LMCHAR SMPLIDNT |

METAIDNT is used to refer to language specific constructs, e.g. *LMCHAR COMMON to refer to the FORTRAN COMMON area; see the individual language guides for which METAIDNTs are supported by the individual language packages

BA := *ESCAPECHAR ('A / 'a) |

this entity has the value of the address of the most recently displayed cell

BB := *ESCAPECHAR ('B / 'b) NUMBER |

this entity has the value of the address at which breakpoint NUMBER is set; it has the value of zero if breakpoint NUMBER is not set

BLN := *ESCAPECHAR ('L / 'l) NUMBER |

this entity has the value of the address of the NUMBER-th local of the current frame

BPN := *ESCAPECHAR ('P / 'p) NUMBER |

this entity has the value of the address of the NUMBER-th formal parameter of the current frame

BQ := *ESCAPECHAR ('Q / 'q)

this entity has the value of the most recently displayed cell

BR := *ESCAPECHAR ('R / 'r)

this entity has the value of the return address for the current frame

BY := *ESCAPECHAR ('Y / 'y)

this entity has the value of the most recently completely evaluated EXPRESSION

Assigning To Address Lists 2e5

Many commands allow the user to assign to an address list as it is being displayed. The specifying of new values to be assigned is handled by the @NVLRUL discussed below.

Two Special Characters ¦ 2f

There are two characters used by DAD as pseudo-interrupts (PSI) that need a separate discussion. The specific characters are initiallized to <CONTROL-L> and <CONTROL-K>, but may be changed by the user by using the Interrupt command.

The first of these characters (initiallized to <CONTROL-L>) is used to get the user to base command mode in DAD. For example, if a user has inadvertently requested DAD to display a large number of cells. Upon realizing the mistake, the user may type 2 <CONTROL-L>s to abort output and return to base command mode. Additionally, when tools are executing (i.e. after the user has given the Continue command), if the user wishes to return to DAD, the user should type one or two <CONTROL-L>s. Since this character is set up as a deferred PSI, it will not take effect until the character is read if only one <CONTROL-L> is typed. If the user wishes immediate action, then 2 <CONTROL-L>s should be typed. (Note that in the case of aborting DAD output it may still take a while until the current contents of the output buffers are empty.)

The second of these special characters (initiallized to <CONTROL-K>) is used to display a short status of tools while they are executing (i.e. after the user has given the Continue command).

Command Summary 3

Debug Command 3a

Overview

The debug command is used to point DAD at a target process. Once DAD is pointed at a target process, the full compliment of DAD commands becomes available.

Syntax

Debug (tool) @TOOLSPEC OK:

TOOLSPEC Rule

If DAD does not know about any tools yet (as when DAD is first started, or after the user has given the Done command for all active tools):

^TENEX-FILE-NAME:

If DAD does know about some processes:

@ACTIVETOOLS:

^IDH:

OPTION ^TENEX-FILE-NAME:

this path allows the user to have one or more parallel processes executing under DAD

ACTIVETOOLS Rule

the FE maintained rule of the usenames for the tools the user is currently debugging

Done Command ¦ 3b

Overview

When the user is done debugging a tool, he/she should issue the Done command. Upon receiving a Done command, DAD will do whatever cleanup is necessary with respect to DAD's knowledge of the tool. If the user was debugging only one tool, or three or more tools, then DAD will ask the user to specify which tool should become the current target tool upon completing the Done command.

Syntax

```
Done (debugging tool) @ACTIVETOOLS OK:
```

ACTIVETOOLS Rule

see the Debug command

Quit Command ¦ 3c

Overview

The Quit command is used to terminate a DAD debugging session and to return the user to the TENEX EXEC.

Syntax

```
Quit (debugging session) OK:
```

Interrupt Command ¦ 3d

Overview

The interrupt command is used to change which characters will serve the 2 special functions of returning to DAD's base command mode and to display the status of executing tools.

Syntax

Interrupt Executing (programs & abort output character should be) ^ICHARACTER OK: ¦

This path allows the user to specify which character will be used to return the user to base command mode.

Interrupt Status (character should be) ^ICHARACTER OK:

This path allows the user to specify which character will be used to display the status of executing tools.

ICHARACTER Selector ¦

Any control character not currently serving any other function.

Wheel Command ¦ 3e

Overview

The Wheel command is used by DAD implementers and maintainers for the debugging and development of DAD. Issuing the Wheel command makes available commands not normally available. The Wheel command requires the knowledge of a special password. It is mentioned here only because it may show up in response to a questionmark (?) typed to see the alternatives available.

Status Command

3f

Overview

The Status commands display the status of the debugger to the user.

Syntax

Status OK:

Status Verbose OK:

This command provides more information about each tool being debugged than the default Status command.

Status For (tool) OK:

This command provides information about the current tool.

Status Verbose For (tool) OK:

This command provides verbose information about the current tool.

Status For (tool) ^IDH OK:

This command provides information for the specified process.

Status Verbose For (tool) ^IDH OK:

This command provides verbose information for the specified process.

Comment.Command ¦ 3g

Overview

This command is used to allow comments to appear on a typescript, etc.

Syntax

```
Comment ^CTEXT:
```

Character Command ¦ 3h

Overview

These commands are used either to display which characters are serving which generic functions, or to modify which character is to serve a specific generic function.

Syntax

Character (set definitions) Display OK:

This command is used to determine which characters are serving which generic functions. Non-standard definitions will appear first in the resulting display.

Character (set definitions) Use ^FCHARACTER (instead of) @CHARRULE OK:

This command is used to change which character will serve a specific generic function.

CHARRULE Rule ¦

"*PLUSCHAR" (for addition):

"*MINUSCHAR" (for subtraction):

"*TIMESCHAR" (for multiplication):

"*DIVIDECHAR" (for division):

"*LPARENCHAR" (for arithemetic groupting left delimiter):

"*RPARENCHAR" (for arithemetic groupting right delimiter):

"*BLOCKCHAR" (for symbol block delimiter):

"*FIELDCHAR" (for record field delimiter):

"*ESCAPECHAR" (for builtin variable escape):

"*LMCHAR" (for language module escape character):

"*SEMICOLONCHAR" (for address list delimiter):

"*COMMACHAR" (for address range delimiter):

"*EQUALCHAR" (for display value):

"*SLASHCHAR" (for display using permanent typeout mode):

"*LSQUARECHAR" (for display numerically):

"*BSLASHCHAR" (for display as a string):

"*RSQUARECHAR" (for display as a record):

"*EXCMARKCHAR" (for display in ascii):

"*QMARKCHAR" (for tell where this symbol is defined):

"*LARROWCHAR" (for assignment):

"*LFCHAR" (for move to next address):

"*UPARROWCHAR" (for move to previous address):

"*TABCHAR" (for move to addressed address):

"*POUNDCHAR" (for move to previously displayed address):

Input Command

¦ 3i

Overview

This command is used to display or change the permanent input mode.

Syntax

```
Input (mode) Display OK:

Input (mode) @INPTYP OK:
```

Typeout Command ¦ 3j

Overview

This command is used to display or change the permanent output mode.

Syntax

```
Typeout (mode) Display OK:

Typeout (mode) @OUTTYP OK:
```

Symbol Command

¦ 3k

Overview

A process may have more than one symbol table. (For example, if different parts of the address space were compiled and loaded as distinct entities.) The symbol commands allow the user to tell the debugger of the location of the symbol tables. When the debugger, and the appropriate Language Module, is first pointed at a process, the LM will use the default location for finding the symbol table.

The debugger makes its own copy of the process' symbol table. Thus, if a process modifies its symbol table, it is necessary for the user to give a new Symbol command. (Ultimately this will be do-able programmatically.)

If a process contains more that one symbol table then the user can point the debugger to different tables by use of the symbol command and the debugger will copy the symbol table the frist time it is pointed to a new location. However, if subsequently, a user points the debugger to a location previously used, the debugger will use its own copy of the symbol table from that location unless the user specifies that there is a new pointer at the location.

Syntax

Symbol (table) Display (status) OK:

This command will display which symbol tables the debugger knows about and will indicate which is the current symbol table and will provide an overview of the current table.

Symbol (table) Pointer (located at) ^SYMADR OK:

Symbol (table) Pointer (located at) ^SYMADR OPTION (undefined symbol table pointer located at) ^SYMADR OK:

These 2 commands will point the debugger to the symbol (and udefined symbol) table(s) at the specified location . If the debugger already has a copy of the symbol table at the specified location, it will not copy the process' table.

Symbol (table) Pointer (located at) OPTION (new pointer at) ^SYMADR OK:

Symbol (table) Pointer (located at) OPTION (new pointer at) ^SYMADR OPTION (undefined symbol table pointer located at) ^SYMADR OK:

These 2 commands will point the debugger to the symbol (and udefined symbol) table(s) at the specified location. This version of the command will force the debugger to make a copy of the specified symbol table(s) regardless of whether or not it already has a copy of the symbol table at the specified location. This is useful if a process has modified its symbol table, or if a process is performing its own swapping in its address space.

Breakpoint Command ¦ 31

Overview

The breakpoint command allows the user to specify that the debugger (conditionally) be entered just prior to the execution of an instruction at a specified address in a target process.

A breakpoint is said to be "hit" when the instruction at the address specified for the breakpoint is about to be executed. After a breakpoint is hit, it either "takes" and the debugger is entered, or it doesn't take and normal execution of the target process continues.

> For each case, i.e. the breakpoint taking or not, the user may specify a string that will be fed to the debugger, as if the user typed it, when the breakpoint is hit.

The decision as to whether or not a breakpoint takes is based on the following algorithm:

> If a user has specified a procedure to be called when a breakpoint is hit, this procedure is called and returns one of three values: take the breakpoint, don't take the breakpoint, or base the decision on the proceed mode and counter. If this procedure returns the third value, or if no procedure was specified, then the breakpoint will take if the proceed mode is normal or automatic or if the proceed mode is count and this breakpoint has been hit count times already without taking.

Every breakpoint that is set, i.e. for which an address has been specified, has the following attributes associated with it:

a) its number, ^BTNUMBER

When a breakpoint is first set, the user can request a specific number, or let the debugger assign an unused number for the breakpoint.

The user uses this number when he or she wishes to modify or examine the status of the breakpoint.

b) its address, ^BTADDRESS

This is the address at which the breakpoint is set.

Note that specifying an address for a breakpoint that is already set is equivalent to first clearing that breakpoint and then setting the address.

c) its name, ^BRNAME

If and when a breakpoint takes, its name will be displayed. A name is simply a string (including the null string) used for information purposes only. If a user is debugging more than one process, he or she may choose to name the breakpoints set in each process with the appropriate process name. Names need not be unique.

d) its proceed mode

Every set breakpoint has one of three proceed modes:

Normal mode

Set either by default or by specify a proceed count of zero.

In this mode, the breakpoint will take each time the breakpoint is hit.

Automatic proceed mode

Set by specifying proceed automatically.

In this mode, the breakpoint will take each time the breakpoint is hit and then the breakpoint will be continued automatically, after processing its take command string if one exists.

Count mode

Set by specifying a non-zero proceed count.

In this mode, the breakpoint will not take until the breakpoint has been hit count plus one times. If a no take command string exists, then the count times this breakpoint is hit before it takes, the no take command string will be executed.

e) its call procedure, ^PNAME - NOT IMPLEMENTED YET

f) its take command string

When a breakpoint takes, if this string is non-null it will be fed to the debugger as if the user entered it on his or her terminal prior to accepting input from the user or automatically continuing.

g) and its no take command string

If a breakpoint doesn't take, and if this string is non-null it will be fed to the debugger as if the user entered it on his or her terminal when the breakpoint is hit and prior to continuing the breakpoint.

Syntax

Breakpoint Display (all) OK:

This command will display the status of all breakpoints that are currently set.

Breakpoint ^BTNUMBER Display OK:

This command will display the status of breakpoint ^BTNUMBER.

Breakpoint Clear (all) OK:

This comand will clear all breakpoints, i.e. make them not set.

Breakpoint ^BTNUMBER Clear OK:

This command will clear breakpoint ^BTNUMBER.

Breakpoint Set (at) ^BTADDRESS @BOPT:

Breakpoint ^BTNUMBER Set (at) ^BTADDRESS @BOPT:

These two commands will set a breakpoint at the specified address, and will set any of the attributes specified. If ^BTNUMBER is not specified, then the debugger will assign a number for this breakpoint. If ^BTNUMBER is specified and it refers to a breakpoint that is already set, then that breakpoint will be cleared first, and then set at the new address with any attributes specified in this instance of the command.

Breakpoint ^BTNUMBER @BOPT1:

This command allows the user to modify the attributes of breakpoint ^BTNUMBER.

BOPT Rule

OK:

@BOPT1:

BOPT1 Rule

Call (procedure) ^PNAME @BOPT:

NOT IMPLEMENTED YET.

This rule is used to specify the name of a procedure that will get called when a breakpoint is hit to determine whether or not to take the breakpoint.

Proceed Count (=) ^PNUMBER @BOPT:

This rule is used to place a breakpoint in either normal proceed mode (if ^PNUMBER is zero) or in count mode.

Proceed Automatically @BOPT:

This rule is used to place a breakpoint in the automatic proceed mode.

Name (for this breakpoint is) ^BRNAME @BOPT:

This rule is used to specify the name for a breakpoint.

Break (commands are) ^BRKCMNDS @BOPT:

This rule is used to specify the take command string that gets executed when a breakpoint takes.

No (break commands are) ^BRKCMNDS @BOPT:

This rule is used to specify the no take command string that will get executed if a breakpoint is hit but doesn't take.

Continue Command

¦ 3m

Overview

The continue commands allow the user to continue the execution of the process(es) that were executing before entering the debugger (regardless of whether the debugger was entered via a (nested) EXEC DEBUG command or by the taking of a (nested) breakpoint), or to modify the address at which a process will have its execution resumed when execution is ultimately continued, and optionally to modify the speed with which execution will proceed.

Syntax

Continue OK:

This command will continue whatever was going_on before the debugger was entered.

Continue OPTION (address for this process is) ^CNADDRESS OK:

This command will change the address at which the current target process will resume execution when it is ultimately continued.

Continue At ^CNADDRESS OK:

This command will change the resume address of the current target process and then continue what was going on before the debugger was entered.

Continue At ^CNADDRESS @CNSPEED OK:

This command will change the resume address of the current target process and then continue what was going on before the debugger was entered, at the specified execution speed.

Continue @CNSPEED OK:

This command will continue what was going on before the debugger was entered, but at the newly speicifed speed.

CNSPEED Rule

Normal (speed):

For (one) @SPDRULE:

Free Command

3n

Overview

Several debugger operations require the use of free cells in the address space of the target process (e.g. breakpoint continuing, executing an instruction on the behalf of the target process). This command allows the user to specify where the debugger should get the cells it requires.

> Currently the debugger requires 4 cells to implement breakpoints. I expect that in the future when I implement instruction execution on the behalf of the target process, specifically procedure calls, the debugger will require as many as 2-3 dozen cells (depending on how many parameters are being passed);

The cells that the debugger is currently using can be determined via the Verbose form of the Status command.

Syntax

```
Free (core available at) ^FCADR OK:
```

Display Command ¦ 3o

Overview

This is the basic command for displaying entities (cells, state information, etc.) in the target process.

Syntax

Display ^DADDRESSLIST:

This command will display the specified address list in the mode specified by the ^DADDRESSLIST terminator (and for certain values of this terminator will let the user modify the displayed address list).

DADDRESSLIST Selector ¦

a ^DADDRESSLIST is an ^ADDRESSLIST that is terminated by either an OK or @DTERM

DTERM Rule ¦

OPTION (typeout mode) @OUTTYP OK:

This terminator will cause the specified address list to be displayed in the output mode specified by @OUTTYP.

OPTION (typeout mode) @OUTTYP OPTION (and assign to address list) OK:

This terminator will cause the specified address list to be displayed in the output mode specified by @OUTTYP, and will allow the user to modify the displayed cells as they are being displayed.

*SLASHCHAR:

*RSQUARECHAR:

*BSLASHCHAR:

*LSQUARECHAR:

*EXCMARKCHAR:

*QMARKCHAR:

*EQUALCHAR:

*LARROWCHAR:

*TABCHAR:

*POUNDCHAR:

*LFCHAR:

*UPARROWCHAR:

Find Command ¦ 3p

Overview

The find commands allow the user to display, and optionally assign to, those cells in an address list that meet certain content requirements. The user may specify a mask to select those bits in a cell that he or she is interested in checking against similar bits in the value that he or she has specified.

(In fact, each cell in the address list is logically ANDed with the mask and the result is then compared with the AND of the mask and the specified search value.)

The mask used in a reference search is one that will select the address field of a cell. The mask used for content and not content searches is the debugger default mask, unless the user specifies a mask for this instance of the command. The default debugger mask can be displayed and modified via the Mask command. It is initially set to select all bits in a cell.

A reference and a content search will display, and optionally allow the user to assign to, those cells in the address list for which the above mentioned compare was equal. A not content search will display, and optionally assign to, those cells that compare unequally.

All displayed cells will be displayed in the current output mode unless the user specified ^FADDRESSLIST terminator modifies the display.

The user may optionally specify an input mode that will be used to evaluate the specified search value, ^FVALUE.

Syntax

Find References (to) @FSPEC (in address list) ^FADDRESSLIST:

This command will display, in the output mode specified by ^FADDRESSLIST terminator, (and, if this terminator dictates it, assign to) those cells in the specified address list whose address field is equal to the specified ^FVALUE.

Find Content @FSPEC (masked by) @MSPEC (in address list) ^FADDRESSLIST:

This command will display, in the output mode specified by ^FADDRESSLIST terminator, (and, if this terminator dictates it, assign to) those cells in the specified address list whose selected bits, as specified by @MSPEC, are equal to the corresponding bits in the specified ^FVALUE.

Find Not (content) @FSPEC (masked by) @MSPEC (in address list) ^FADDRESSLIST:

This command will display, in the output mode specified by ^FADDRESSLIST terminator, (and, if this terminator dictates it, assign to) those cells in the specified address list whose selected bits, as specified my @MSPEC, are not equal to the corresponding bits in the specified ^FVALUE.

FSPEC Rule ¦

^FVALUE:

the search value

OPTION (input mode) @INPTYP (value) ^FVALUE:

This rule allows the user to specify a current input mode that will be used to evaluate ^FVALUE and ^MVALUE

MSPEC Rule ¦

OK:

Use the default debugger mask

^MVALUE:

The mask to be used for this instance of the find command.

FADDRESSLIST Selector ¦

a ^FADDRESSLIST is an ^ADDRESSLIST that is terminated either with an OK or @FTERM

FTERM Rule

*SLASHCHAR:

*RSQUARECHAR:

*EXCMARKCHAR:

OPTION @OUTTYP OK:

OPTION @OUTTYP OPTION (and assign to address list) OK:

Mask Command ¦ 3q

Overview

This command allows the user to examine or to modify the default debugger mask, which is used by the Find and Memory commands.

Syntax

```
Mask Display OK:

Mask Set (to) ^MVALUE OK:

Mask Set (to) OPTION (input mode) @INPTYP (mask value)
^MVALUE OK:
```

Memory Command ¦ 3r

Overview

The memory commands allow the user to set (selected bits) in all cells in the specified address list to the specified value.

If the user does not specify to use a mask, then all bits in the pertinent cells will be affected. If the user specifies to use a mask, then he or she may use either the default debugger mask, or may specify a mask for this instance of the command.

If a mask is used then only those bits selected by the mask will be set, and they will be set to the corresponding bits in the specified ^MNVALUE.

Syntax

Memory (set to) @MNSPEC (in address list) ^MADDRESSLIST:

This command will set the selected bits in the cells in the specified address list to the corresponding bits in the specified ^MNVALUE.

MNSPEC Rule ¦

^MNVALUE:

the value to set the selected bits to

OPTION (input mode) @INPTYP (value) ^MVALUE:

this path allows the user to specify a current input mode that will be used to evaluate ^MNVALUE, and ^MVALUE (if one is specified)

MADDRESSLIST Selector ¦

a ^MADDRESSLIST is an ^ADDRESSLIST is terminated by either an OK or @MTERM

MTERM Rule ¦

OPTION (masked by) OK:

this path indicates to use the default debugger mask to select bits in the address list for modification

OPTION (masked by) ^MVALUE OK:

this path allows the user to specify a mask to use to select bits in the address list to be modified

Output Command ¦ 3s

Overview

The output commands give the user the capability to multiplex output to his or her terminal and/or to a sequential text file. If output is currently going only to a file, the user will not have the ability to modify cells in an address list unless the Type command is used. If output is going only to a terminal, the user can force output to a file by use of the Print command. When the user first specifies a file, output will be sent to both the file and the terminal. When specifying a file, the user can either specify a new file, or an old file to which the output should be appended.

(Current version of the debugger uses local TENEX files for the output file, but ultimately this will be a WM file.)

Syntax

```
Output (printing) Display:

Output (printing) Append (to file) ^OLDFILELINK OK:

Output (printing) To (file) ^NEWFILELINK OK:

Output (printing) Off OK:

Output (printing) Both (to file and terminal) OK:

Output (printing) Soley (to) File (and not to terminal)
OK:

Output (printing) Soley (to) Terminal (and not to file)
OK:
```

Print Command

3t

Overview

This command is used to display the specified address list on the specified file. If there is already a specified output file, then this is the one that will be used, and the user will not be asked to specify a file.

(Note that when using this command, it is not possible to modify the cells as they are being displayed since they will be displayed on an output file and not on hte user's terminal. Useful for core dumps, among other uses!)

Syntax

Print ^PADDRESSLIST:

This command will display the specified address list on the current output file in the mode specified by ^PADDRESSLIST terminator.

Print (on file) ^NEWFILELINK ^PADDRESSLIST:

This command will display the specified address list on the specified output file in the mode specified by ^PADDRESSLIST terminator.

PADDRESSLIST Selector

a ^PADDRESSLIST is an ^ADDRESLIST terminated by either an OK or @PTERM

PTERM Rule

OPTION (typeout mode) @OUTTYP OK:

*SLASHCHAR:

*RSQUARECHAR:

*BSLASHCHAR:

*LSQUARECHAR:

*EXCMARKCHAR:

*QMARKCHAR:

*EQUALCHAR:

*TABCHAR:

*POUNDCHAR:

*LFCHAR:

*UPARROWCHAR:

Type Command ¦ 3u

Overview

This command is used to display the specified address list (in the specified mode) on the user's terminal regardless of his output file settings.

Syntax

```
Type ^DADDRESSLIST:

   (See the Display command.)
```

Value Command ¦ 3v

Overview

This command is equivalent to:

```
Display ^ADDRESSLIST *EQUALCHAR:
```

Syntax

```
Value (of) ^VADDRESSLIST:
```

VADDRESSLIST SELECTOR

a ^VADDRESSLIST is an ^ADDRESSLIST terminated by either an OK or @VTERM

VTERM Rule ¦

```
*EQUALCHAR:
```

Speed Command ¦ 3w

Overview

The speed command allows the user to modify the execution speed of the current process. The execution speed can be modified so that the process will execute in a single step mode (a single machine or language instruction at a time); and/or to treat an entire called procedure as if it were one instruction; and/or to execute until a branch or transfer instruction is encountered; and/or to continue automatically after entering the debugger and notifying the user because one of the above conditions has been met.

Syntax

Speed (of execution) Normal OK:

This command resets the execution speed for the current process back to normal speed.

Speed (of execution) Single @SPDRULE:

This command allows the user to modify the execution of the current process.

*BSLASHCHAR Command ¦ 3x

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Z *BSLASHCHAR:
```

Syntax

```
"*BSLASHCHAR":
```

*EQUALCHAR Command ¦ 3y

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Z *EQUALCHAR:
```

Syntax

```
"*EQUALCHAR":
```

*EXCMARKCHAR Command ¦ 3z

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Z *EXCMARKCHAR:
```

Syntax

```
"*EXCMARKCHAR":
```

*LSQUARECHAR Command 3aa

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Z *LSQUARECHAR:
```

Syntax

```
"*LSQUARECHAR":
```

*RSQUARECHAR Command 3ab

Overview

This command is equivalent to:

Display *ESCAPECHAR Z *RSQUARECHAR:

Syntax

"*RSQUARECHAR":

*QMARKCHAR Command 3ac

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Z *QMARKCHAR:
```

Syntax

```
"*QMARKCHAR":
```

*SLASHCHAR Command

3ad

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Z *SLASHCHAR:
```

Syntax

```
"*SLASHCHAR":
```

*LARROWCHAR Command

3ae

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Z *LARROWCHAR:
```

Syntax

```
"*LARROWCHAR":
```

*UPARROWCHAR Command

3af

Overview

This command is equivalent to:

```
Display *ESCAPECHAR A - 1:
```

Syntax

```
"*UPARROWCHAR":
```

*LFCHAR Command

3ag

Overview

This command is equivalent to:

```
Display *ESCAPECHAR A + 1:
```

Syntax

```
"*LFCHAR":
```

*TABCHAR Command

3ah

Overview

This command is equivalent to:

```
Display *ESCAPECHAR Q:
```

Syntax

```
"*TABCHAR":
```

*POUNDCHAR Command

3ai

Overview

This command is the inverse for the last Tab, Linefeed, Uparrow, or Pound command.

Syntax

"*POUNDCHAR":

Common Rules

3aj

BASE Rule

Decimal:

Octal:

Hex:

Binary:

INPTYP Rule

Ascii:

Bytes (with bytesize of) ^BSVALUE:

Floating (point numbers):

Rad50:

Radix ^RXVALUE:

Radix @BASE:

Sixbit:

Symbolic:

OUTTYP Rule

Addresses (as) Absolute (values):

Addresses (as) Symbolic (values):

Array:

Ascii:

Bytes (with bytesize of) ^BSVALUE:

Floating (point numbers):

List:

Numeric:

Rad50:

Radix ^RXVALUE:

Radix @BASE:

Records (of name) ^RNAME:

Sixbit:

String:

Symbolic:

NVLRUL Rule |

CD: |

abort the display of, and assignment to, this address list

OK:

accept the displayed value of this entity

^NVALUE OK:

replace the value of the displayed entity with ^NVALUE, which will be interpretted according to the current input mode

OPTION (input mode) @INPTYP (new value) ^NVALUE OK:

replace the value of the displayed entity with ^NVALUE, which will be interpretted according to the specified input mode

@MOVRUL:

^NVALUE @MOVRUL:

OPTION (input mode) @INPTYP (new value) ^NVALUE @MOVRUL:

the above 3 paths allow the user to terminate the (optionally) newly specified value (for the displayed entity) with the @MOVRUL paths. When the display of, and assignment to, the specified address list is finished, the last specified @MOVRUL path will take effect as if the user had given the GFC command corresponding the the @MOVRUL path

MOVRUL Rule

*TABCHAR:

*POUNDCHAR:

*LFCHAR:

*UPARROWCHAR:

SPDRULE Rule

In the following, the ordering of @SPDPROC, @SPDEXEC, and @SPDCONT, is not important, and 0, 1, 2, or all 3 of the rules may appear.

@SPDTYPE OK:

@SPDTYPE @SPDPROC OK:

@SPDTYPE @SPDEXEC OK:

@SPDTYPE @SPDCONT OK:

@SPDTYPE @SPDPROC @SPDEXEC OK:

@SPDTYPE @SPDPROC @SPDCONT OK:

@SPDTYPE @SPDEXEC @SPDCONT OK:

@SPDTYPE @SPDPROC @SPDEXEC @SPDCONT OK:

SPDCONT Rule

Proceed (automatically after each instruction):

This rule allows the user to continue automatically after entering the debugger because some single stepping mode condtion has been met.

SPDEXEC Rule

Execute (until branch point or transfer instruction):

This rule allows the user to have execution of the process continue until a branch or transfer instruction is encountered.

SPDPROC Rule

Treat (called procedures as one instruction):

This rule allows tge user to treat a called procedure as one instruction rather than as a series of instructions.

SPDTYPE Rule

Language (instruction):

This rule means to deal with instructions at the high level language level as opposed to at the machine level.

Machine (instruction):

This rule means to deal with instructions at the machine level as opposed to at the high level language level.

Selectors

3ak

In the following discussion an expression is really a text selector that conforms to the rules for expression generation for the current language being used by the debugger.

ADDRESSLIST Selector

text that conforms to the formal definition of an address list (see above)

BRKCMNDS Selector

any text

BRNAME Selector

any text

BSVALUE Selector

a number in the current input mode radix

BTADDRESS Selector

an expression that evaluates to an address

BTNUMBER Selector

a number in the current input mode radix

CNADDRESS Selector

an expression that evaluates to an address

CTEXT Selector

any text

FCADR Selector

a number in the current input mode radix

FCHARACTER Selector

a single non-alphanumeric character

FVALUE Selector

an expression

IDH Selector

a number in the current input mode radix

MNVALUE Selector

an expression

MVALUE Selector

an expression

NEWFILELINK Selector

an new file name string

NVALUE Selector

any text

OLDFILELINK Selector

an old (pre-existing) file name string

PNAME Selector

an expression that evaluates to an address

PNUMBER Selector

a number in the current input mode radix

RNAME Selector

an expression that evaluates to the address of a record descriptor

RXVALUE Selector

a base ten number

SYMADR Selector

a number in the current input mode radix

TENEX-FILE-NAME Selector

a full TENEX file name

Appendix I - L10 'Cheat' Card

¦ 4

Appendix II - Alphabetical List of Commands, Rules, and Selectors 5

Commands

```
Find Commands ------------------------------------- page 43
   Find Content @FSPEC (masked by) @MSPEC
            (in address list) ^FADDRESSLIST:
   Find Not (content) @FSPEC (masked by) @MSPEC
            (in address list) ^FADDRESSLIST:
   Find References (to) @FSPEC
            (in address list) ^FADDRESSLIST:

Free (core available at) ^FCADR OK: --------------- page 40

Input Commands ------------------------------------ page 29
   Input (mode) Display OK:
   Input (mode) @INPTYP OK:

Interrupt Commands -------------------------------- page 23
   Interrupt Status (character should be) ^ICHARACTER OK:
   Interrupt Executing (programs & abort output character
            should be) ^ICHARACTER OK:

Mask Commands ------------------------------------- page 46
   Mask Display OK:
   Mask Set (to) ^MVALUE OK:
   Mask Set (to) OPTION (input mode) @INPTYP
            (mask value) ^MVALUE OK:

Memory (set to) @MNSPEC (in address list)
        ^MADDRESSLIST: ------------------------- page 47

Output Commands ----------------------------------- page 49
   Output (printing) Off OK:
   Output (printing) Display:
   Output (printing) To (file) ^NEWFILELINK OK:
   Output (printing) Append (to file) ^OLDFILELINK OK:
   Output (printing) Both (to file and terminal) OK:
   Output (printing) Soley (to) File (and not to terminal)
            OK:
   Output (printing) Soley (to) Terminal (and not to file)
            OK:

Print Commands ------------------------------------ page 50
   Print ^PADDRESSLIST:
   Print (on file) ^NEWFILELINK ^PADDRESSLIST:

Quit (debugging session) OK: ---------------------- page 22
```

Rules

Selectors