NAME

ddt − debug remote and local programs

SYNOPSIS

ddt [**−cdfhinrstuw**] [*file* [*file*]]

DESCRIPTION

Ddt is an interactive symbolic debugger that allows remote and local debugging at the assembly language level. Locally, *ddt* works through *ptrace*(2) by default or through a /dev/mem file. Remotely, *ddt* communicates through a serial link. Code may be displayed and breakpoints set. Single-stepping is possible at machine instruction level, procedure level, or on non-sequential instruction fetch. *Ddt* supports debugging in physical address space, supervisor virtual address space, and user virtual address space.

When no options are specified and two file names are given, the first file must be an *a.out* format file with symbols and the second file must be a core file. To specify a core file without specifying an *a.out* file, use the **−c** option.

Ddt also allows examination of:

> Vmunix core files (produced with *savecore*(8)) using the **−u** option.
>
> Vmunix through /dev/mem or /dev/kmem using the **−f** option.
>
> Any GENIX file without symbols (using **−f** *file*).

Options are:

−c Indicate that the file is a core file. No symbols are availiable. Used only when one file name is given.

−d Set input and output radix to decimal.

−f Examine *file* “straight.” No symbols exist and addresses represent actual offsets into the file.

When the **−f** option is specified with two filenames, *ddt* assumes the first is a file in *a.out* format and the second is a /dev/mem file. *Ddt* examines memory through the /dev/mem files, not through the system *ptrace* calls. Symbols are available in this case.

−h Set input and output radix to hexidecimal.

−i Open *file,* but do not allow it to be executed. **−i** is useful for comparing and checking a.out files. It cannot be used with **−c** or **−f.**

−n Do not use the monitor commands that check the NS16082 Memory Management Unit (MMU) for address translation. This option is for remote debugging ONLY. It tells *ddt* that the remote system does not have an MMU. *Ddt* displays all addresses as virtual (as the CPU sees addresses). NOTE: If an MMU is present in the remote system and enabled for translation, this option will cause aborts to be caught by ABORT vector (NS16032) of either *ddt* or the last NS16000 program to set that vector.

−r Debug remotely. This allows debugging on a remote NS16000 board. The remote board must be set up as described in *GENIX Cross-Support Software for an NS16000-based System.*

−s Dump all program symbols in the *a.out* file on standard output, then exit. In this symbol dump, *ddt* lists each symbol followed by the hex value for that symbol and its module table address (in decimal). Even though all the symbols are shown, where symbol name or symbol value conflicts exist (i.e., two or more symbols with same value or name), either the external symbol will replace the local or the last found symbol will replace any other.

−t Scan the Module Table for symbols. Normally, *ddt* takes the module number in the symbol table entry *stab.h* when it computes module table-related instruction fields (sb relative, external). This option finds the module number by scanning the Module Table for the best match. This is only useful for tracking link errors.

−u vmunix vmcore.N
Debug vmunix core. *Ddt* examines a vmunix core file (made by *savecore*(8)) with the corresponding vmunix and vmsymbols in /. The panic string will be printed for the crash. Registers at time of panic can be looked at, addresses in the kernel will be translated (i.e., for looking at u. area).

−w Write enable *file*. **−w** must be used with **−f or −i**. Using **−fw** allows *file* to be changed. Using **−iw** allows patching on *a.out* files. **−w** is useful for changing code by hand.

When no options are specified, *ddt* opens an *a.out* file in the current directory in hexidecimal mode for local execution.

In the following descriptions, the following characters are specified symbolically:

$ = escape

<cr> = return

<lf> = line feed (ˆj or ')

<bs> = backspace

Ddt Command Format

Most command lines use the format:

[*expression*] ... [**$** [**$**]] [*number*] ... *command* [**<cr>**]

Command is a command character or characters. *Expression* and *number* arguments, such as symbols defined in an *a.out* file and numbers in the current radix, can optionally be used with certain command characters. Most commands are specified with one or two escape characters (**$**). The typical command consists of "**$***command*", which is escape followed by a command character. Some commands have multiple command characters and some require a **<cr>** to complete.

Commands are executed immediately. When symbols or expressions are typed, <bs> may be used for input editing.

Symbols

Symbols are composed alphanumerics, underlines (_), dollar signs ($), and periods (.).

The "$" and "." symbols are defined by *ddt.* Alone, the symbol "." refers to the most recently specified address, or if it is used immediately after a number, "." means input the number in decimal radix. *Ddt* translates "$" into ".", so "$" functions the same as ".".

Symbols other than "$" and "." are defined by the file being debugged.

All NS16032 CPU registers are available; they are specified to *ddt* by the register names:

r0, r1, r2, r3, r4, r5, r6, r7	General Purpose Registers 0-7
psr	Processor Status Register
f0, f1, f2, f3, f4, f5, f6, f7	Floating Point Registers 0-7 (local mode only)
mod	Module Table Register
sb	Static Base Register
pc	Program Counter
sp1	User Stack Pointer
sp0	Interrupt Stack Pointer
fp	Frame Pointer
intbase	Interrupt Base Register

These names are the same as those given in the NS16032 data sheet.

When a symbol is specified in a command line, the symbol can be abbreviated and an initial underline (_) need not be typed. When *ddt* looks for the symbol it selects the best match. For example, assuming the symbol "XYZ" has been typed as part of a command line, *ddt* will look through all the symbols and select the one closest to "XYZ", as follows:

1) XYZ
2) _ XYZ
3) XYZ$*anything*
4) XYZ*anything*
5) _ XYZ*anything*

If *ddt* does not find any of the six possible matches, it prints “symbol <XYZ> not found”. If *ddt* returns, for example, “XYZTOMATO”, then the symbols “XYZ” and “_ XYZ” do not exist in the file; the symbols “_ XYZ*anything*”, “XYZ*anything*”, and “_ XYZ*anything*” may or may not be present in the file. *Anything* field will match the first matching symbol.

Expressions

Expressions are composed of symbols, numbers, and operators.

When *ddt* evaluates expressions, it finds the value of primary expressions first. (Primary expressions, for example, symbols and numbers, have intrinsic values.) Next, *ddt* negates or complements the primary expression. *Ddt* performs multiply, and, div, mod, shift right, and shift left operations. Last, *ddt* does add, subtract, or, and xor operations in the expression. The following details *ddt*'s order of evaluation for expression:

Low Priority

expr ::= D1 “+” D1	add
| D1 “-” D1	subtract
| D1 “|” D1	or
| D1 “^” D1	xor

Medium Priority

D1 ::= D2 “*” D2	multiply
| D2 “&” D2	and
| D2 “#” D2	div
| D2 “%” D2	mod
| D2 “>” D2	shift right by second D2
| D2 “<” D2	shift left by second D2

High Priority Operators

D2 ::= “-” D3	negate
“~ ” D3	complement

Primary Expressions

D3 ::= “(” expr “)”	subxpression
“ . ”	dot
“ ‘ ”	last displayed value
D3 "@e	indirect through D3
symbol	take symbol value;

%symbol	assume % is part of symbol
number [0-9,a-f,A-F]	if leading digit a-f provide leading 0, for example, type “0a” for hex a
number[.,o,x]	. means decimal, o octal, and x hex
register[r0-r7,fp,sp,pc,psr,mod,msr,eia]	

And, for remote debugging add:

register[bpr0,bpr1,bcnt,pf0,pf1,sc0,ptb0,ptb1,intb]

The postfix ‘@’ operator makes the expression a pointer and the value is the 4-byte quantity at that memory address.

When a register is used in an expression, it is both an address and a value. For example, “r0+4” is the contents of r0 plus 4, while “r0/” is the contents of r0 and “r0/5<cr>” stores 5 in r0. “r0/<lf><lf>” prints the contents of r0, r1, and r2. (Be careful, “r0<lf>” stores the contents of r0 in the currently open location.)

Examples:

Consider the addressing mode 4(8(fp)). The effective operand address could be displayed with:

```
fp+8@+4=
```

or:

```
fp+8@+4;
```

The operand could be displayed in the current mode with:

```
fp+8@+4/
```

The more complicated case of “4(8(fp))[r2:w]” can be displayed with: “fp+8@+4+r2*2i” (then “=”, “;”, “/”, etc.)

For ext(a)+b, type:

```
mod+4@+a*4@+b
```

Mode Selection Commands

Mode selection commands tell *ddt* what format to use for displaying output. The output format mode is specified locally or permanently. Typing a single **$** changes the output mode locally; typing **$$** changes the output mode permanently. Local formats remain in effect until the next **<cr>** is typed. Permanent formats are effective until another permanent mode is specified.

Mode selection commands are:

$mb Select absolute numeric mode. Print the numeric value using the current radix. $mb is the same as **$mn** , except that **$mb** ignores **$mA** and **$mr .**

$mc Select character mode. Show the hexadecimal value of nonprintable characters preceded by a backslash (/).

$mf Select floating-point mode. Print numbers as floating-point numbers.

$mi Select instruction mode. Display memory as assembly instruction mnemonics and their operands. (**$mi** mode is used by automatic mode (**$mA**) when the address is in the program code area.)

$mn Select numeric mode. **$mn** is the same as **$mb**, but **$mr** and **$mA** override **$mn**.

$ms Select string mode. Show memory contents as character strings, and stop printing on null.

$mA Select automatic mode (default except for **−f** *file* option). *Ddt* prints the data at addresses in the program code area as instructions (like under “**$mi**”); *ddt* prints the data at other addresses numerically, according to the size given by “$tX”.

$mN Select normal mode (default for **−f** *file* option). **$mN** prints all data numerically and no symbols are available.

$ma Select nonsymbolic mode.

$mr Select symbolic mode. **$mr** overrules **$mn**.

*value***$mm**
Set maximum offset. The maximum offset tells *ddt* to show addresses as offsets from a symbol until the offset is greater than *value*. Default *value* is 1000.

$r[**bodx**]
Set input/output radix as in printf: b=binary, o=octal, d=decimal, x=hex. Character or instruction input modes are not available.

$t[**bwd**]
Select the context or data size: b=byte, w=word, d=doubleword. Default is **d**.

$n Like "=" only display numeric as unsigned.

A decimal radix number can always be entered regardless of the input radix by typing the number followed by a "."; i.e., "234." is 234 base 10. Likewise a hexadecimal radix number can always be entered by typing the number followed by a "x"; i.e., "2aex" is 2AE base 16. An octal radix number can always be entered by typing the number followed by an "o".

Run Commands

The run commands are:

$g and **$G** *arguments* **<cr>**
Begin execution of the *a.out* (or *file*). **$G** allows arguments to be passed to the program through *ddt*. For example, to debug "/usr/ucb/ls −R", give the run command "$G −R <cr>". Arguments need only be set once with **$G**; on repeat runs, **$g** will use the same arguments. **$G** can be issued at any time to change arguments.

When *ddt* begins execution, it gives the name of *file* and the arguments; for example, *ddt* prints:

```
running /usr/ucb/ls −R
```

Both **$g** and **$G** put all breakpoints in a program before running.

For remote debugging, **$g** and **$p** are the same, except **$g** will not proceed from a breakpoint. **$G** does not pass arguments in remote mode.

Retyping Output

Ddt regcognizes three commands for retyping output:

;	Retype the last value in symbolic format.
=	Retype the last value as a number.
$n	Retype the last value as a unsigned number.

Display (or Open Location) Commands

These commands display code or data and "open" locations.

In the following table, "the new address" means the last value typed, either by *ddt* or through the keyboard. However, "if typed" means "if typed on the keyboard." Certain commands change the location counter (.). Open location commands and their effects on "." are:

Command:	Function:	Changes “.”?
!	Open the new address.	If typed.
/	Open and type the new address.	If typed.
\	Open and type the new address.	Never.
ˆI (tab)	Open and type the new address.	Always.

For example, “10/” displays address 10 and the contents of address “10”; it leaves “.” set to 10. “10\” displays the same things, but it does not change “.” at all.

Display or Change Commands

These commands display and change memory locations. To change the contents of a location, the address must be open (see previous section) and replacement expression must be specified in the command line. *Ddt* stores the value of the replacement expression in the location.

Some commands increment or decrement the location counter (.). The delta depends on the contexts established with the **$t***X* command, or in **$mi** mode, it depends on the size of the instruction.

All change or display commands open the new “.”.

The change or display commands are:

?	Decrement “.” and display the new address.
*expr***?**	Store *expr* first then decrement “.” and display the new address.
< lf>	Increment “.” and displays the new address.
*expr***’**	Store *expr* then decrement “.” and display the new address.
*expr***< lf>**	Store *expr* then decrement “.” and display the new address.
<cr>	Cancel temporary output format modes.
*expr***<cr>**	Store *expr* then decrement “.” display the new address.

Program Control Commands

Program control commands manage breakpoints, step through a program, provide a help facility, quit *ddt*, display memory, and create command strings. For these commands the radix of *number* is always decimal.

addr$**b** Set breakpoint at *addr*. If *addr* is not specified, then the value of “.” is used.

$*number***b**
Remove breakpoint *number.*

$B Remove all breakpoints.

$l List all existing breakpoints. For remote debugging, also list which address space breakpoint are in (user or supervisor).

*number***$p**
Proceed from current pc, with *number* proceeds from breakpoints; default is 0. Proceed is done by stepping one instruction, (a “ **[**”), inserting all breakpoints and running.

*number***]**
Single-step pc. Do not insert breakpoint. Default is 0.

*number***[**
Single-step over pc. If the next instruction is cxp, cxpd, jsr or bsr, step over call. Also, for remote debugging step over rett, reti. Does not insert breakpoint. *number*[will do *number* single-steps. Default is 0.

} Single-step. Insert breakpoints.

{ Single-step pc. If the instruction is cxp, cxpd, jsr or bsr, step over call. Also, for remote debugging, step over rett, reti. Insert breakpoints.

$k Skip the current instruction.

$s Display a stack trace. Show offset in function and arguments to call (for C call-return sequences).

$S Display a stack trace. Show offset in function but no arguments to call (for bad stacks and non-C stacks).

$U Put an uplevel breakpoint at the return pc of the next frame.

$u Insert an uplevel breakpoint at the return pc of the next frame and proceed. Execution proceeds to that breakpoint because **$u** causes *ddt* to temporarily ignore any intervening breakpoints. When *ddt* reaches the next frame, **$u** removes the breakpoint it inserted. **$u** differs from **$U** in that it proceeds directly to the next frame. Note: **$u** can be used to get out of a call. To work as intended, be sure to step past the ENTER instruction in the current routine (the fp hasn't changed yet). If you are not beyond the ENTER, then the breakpoint will be placed at a depth one greater than expected.

*calladdr, argn... arg1***$c**
Perform a cxp call to the address given. The address must be the start of routine and have the correct module value for that symbol. The arguments will be pushed on the stack in right to left order. If the program stops due to a breakpoint within the call routine, the arguments will not be cleaned off the stack on proceed or step.

$h Print a synopsis of *ddt* commands.

$q Quit *ddt*.

^d Display next 10 data items.

^b Single-step 10 times (10 ‘ **]** ’).

^f Single-step over 10 times (10 ‘ **[** ’).

*number***$e***command-string*
Execute the *command-string* when breakpoint *number* is hit, or if no *number* is given, whenever the program stops. A *command-string* is any other *ddt* command. For example: “$er0/” will show the contents of r0 every time the program stops (single-step or breakpoint).

*number***$E**
Disable command string for breakpoint *number*, or if no number is given, disable the **$e** (the any stop string). **$***number***b** will also disable command string *number*, and **$B** will disable all *number* command strings.

$L List all stop strings. **T**: is the any break command string. Stop strings are set with **$e**. **“1:”** would indicate breakpoint number 1 has that command string.

Remote Debugging Commands

This section describes commands available only during remote debugging.

$d Download the *a.out* file over the serial line. When *ddt* transfers a file, it prints to standard output the number of bytes remaining to be loaded (in increments of a thousand).

$v Switch to supervisor mode examination. The sp register is now sp0. The address space is as defined by the Memory Management Unit, MMU, (NS16082) for supervisor mode.

$V Switch to user mode examination. The sp register is now sp1. The address space is as defined by the MMU (NS16082) for user mode.

$P Nonsequential step. Progress until one nonsequential trap (as defined by MMU) has occurred.

*addr***$x** Set an MMU breakpoint at address or “.” if no address is given. Only one MMU breakpoint may be set. It can be seen with **$l** and removed with **$***number***b** where *number* is the number given by **$l**. When the breakpoint occurs, *ddt* prints a message stating the kind of access that caused the break. **$x** will break on execute, read, and write.

*addr***$xx**
Set an MMU breakpoint; same as above only break on execute.

*addr***$r** Set an MMU breakpoint; same as above only break on read.

*addr***$xw**
Set an MMU breakpoint; same as above only break on write.

*addr***$xR**
Set an MMU breakpoint; same as above only break on read or write.

Signals

When using *ddt* locally, signals sent to the program being debugged are shown with standard GENIX names (see /usr/include/signal.h). Remotely *ddt* shows traps caught by the running program with NS16000 trap names. For example:

Trap(UND)
An undefined instruction trap occurred.

FILES

/dev/tty?? the line used for remote debugging

SEE ALSO

cu16(1), ptrace(2)

BUGS

The *ptrace*(2) addr argument for read or write to the u area (user process data structure within the kernel) is nonstandard. If the value matches any for registers in <sys/reg.h>, the request refers to a register. Otherwise, it is an index into the u.