B

The NetWare Debugger

Introduction

The NetWare operating system includes an internal assembly language oriented debug utility. The NetWare debugger allows a developer to perform the commands summarized in the following table. These commands and examples of their use are explained in the remainder of this appendix.

.A	displays the abend or break reason
B	displays all current breakpoints
BC number	clears the specified breakpoint
BCA	clears all breakpoints
B = addr [condition]	sets an execution breakpoint at address
BW = addr [condition]	sets a write breakpoint at address
BR = addr [condition]	sets a read/write breakpoint at address
C addr	changes memory in interactive mode
C addr=number(s)	changes memory to the specified number(s)
C addr="text"	changes memory to the specified text ASCII values
.C	does a diagnostic memory dump to diskette
D addr [length]	dumps memory for optional length
DL[+linkoffset] addr [length]	dumps memory starting at address for optional length and traverses a linked list (default link field offset is 0)
REG=value	changes the specified register to the new value REG is EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP, EIP, or EFL
F Flag=value	changes the FLAG bit to value (0 or 1) where FLAG is CF, AF, ZF, SF, IF, TF, PF, DF or OF
G [break addr(s)]	begins execution at current EIP and set optional temporary breakpoints(s)
H	displays basic debugger command help screen
HB	displays breakpoint help screen
HE	displays expression help screen
.H	displays the dot help screen
I [B;W;D] Port	inputs byte, word, or dword from Port (default is byte)
M addr [L length] pattern	searches memory for pattern (L length is optional and if not specified, the rest of memory will be searched)
.M	displays loaded module names and addresses
N symbolName addr	defines a new symbol name at address
N -symbolName	removes defined symbol name
N--	removes all defined symbols
O [B;W;D] Port=value	outputs byte, word, or dword value to PORT
P	proceeds over the next instruction
.P	displays all process names and addresses
.P addr	displays <address> as a process control block
Q	quits and exits back to DOS
R	displays registers and flags
.R	displays the running process control block
S	single-steps
.S	displays all screen names and addresses
.S addr	displays <address> as a screen structure
T	trace (single-step)
U addr [count]	unassembles count instructions starting at address
V	views server screens
.V	displays server version
Z expression	evaluates the expression (See HE help screen)
? [addr]	If symbolic information has been loaded, the closest symbols to address (default is EIP) are displayed

Invoking the Debugger

There are four methods available to invoke the debugger.

From the server console keyboard

1) Press the **Ctrl - Alt - LeftShift - RightShift - Esc** key combination simultaneously at the server console keyboard. This will not work if the server is hung in an infinite loop with interrupts disabled or if the server console is secured.

2) After the driver abends or GPIs the server, enter the key combination described in method 1 above or type **386debug**. The characters do not echo to the screen, but the debugger prompt (**#**) appears.

From a driver or NLM

3) Include an **INT 3** in the desired code segment where the breakpoint is to be executed. Programs written in C using CLIB can call the *Breakpoint ( )* function. Programs written in C using the OS library can call the *EnterDebugger ( )* function.

Manually

4) Generate a non-maskable interrupt with an NMI board. This will cause the server to Abend, after which method 2 above may be performed. This method may be required if the software being debugged is in an infinite loop with interrupts disabled.

When the debugger is entered, it will display the location at which the trap occurred, the cause of the trap into the debugger, and the contents of the general registers and flags.

Once you have entered the debugger, the address and length of the data and code segments of all loaded modules may be found using the **.m** command. Breakpoints can then be set in the driver code using addresses in the map file relative to the addresses dumped by the debugger.

The available debugger commands are explained on the following pages of this appendix.

Debug Commands

Help

The debugger's help commands are:

H	display help for general commands
HB	display help for breakpoints
HE	display help for expressions
.H	display help for "." commands

"." Commands

.a	display the Abend or break reason
.c	do a diagnostic memory dump to diskette
.h	display help for "." commands.
.m	display loaded module names and addresses.
.p	display all process names and addresses.
.p *addr*	display *address* as a process control block
.r	display running process control block.
.s	display all screen names and addresses.
.s *addr*	display all screen names and addresses.
.v	display server version

Breakpoints

There are four breakpoint registers, allowing a maximum of four breakpoints to be set at the same time. The breakpoints can be permanent breakpoints, set using the B commands (described in this section), or temporary breakpoints set using the G command. In addition, the P command will also set a temporary breakpoint if the current instruction cannot be single stepped. This section consists of descriptions and examples for setting permanent breakpoints. Temporary breakpoints using the G and P commands are described later in this chapter.

Breakpoint Conditions

Several breakpoint commands include an optional *[condition]* argument. A breakpoint condition is any expression to be evaluated when the break occurs. If the condition is false, execution is resumed immediately without entering the interactive debugger.

B

Display all breakpoints that are currently set.

```
# B

Breakpoint 0 write byte at FFF65623
Breakpoint 1 read or write byte at 000653BA
Breakpoint 2 execute at FFF06BA3
```

BC *number*

Clear the breakpoint specified by *number*.

```
# BC 2

Breakpoint cleared
```

BCA

Clear all breakpoints.

```
# BCA

All breakpoints cleared
```

B = *address [condition]*

Set an Execution Breakpoint at the *address* specified when the indicated *[condition]* is true.

```
# B = FFF8765A

Set as breakpoint 0
```

BW = *address [condition]*

Set a Write Breakpoint at the *address* specified when the indicated *[condition]* is true.

```
# BW = FFF665AB

Set as breakpoint 1
```

BR = *address [condition]*

Set a Read/Write Breakpoint at the *address* specified when the indicated *[condition]* is true.

```
# BR = 0065ACF3

Set as breakpoint 2
```

Memory

This section describes how to change or display memory contents.

C *address*

Interactively change the contents of memory location *address*.
(To end interactive mode type a period.)

```
# C FFF6432A

FFF6432A (00)=33
FFF6432B (34)=C8
FFF6432C (5A)=.
```

C *address = number(s)*

Change the memory contents beginning at *address* to the specified *number(s)*.

```
# C FFF534C5 = 00,00,12,5A,78

Change successfully completed
```

C *address = "text string"*

Change the memory contents beginning at *address* to the specified *text string*.

```
# C FFF60DB3 = "This is a string."

Change successfully completed
```

D *address [count]*

Dumps the contents of memory, starting at *address*, for *[count]* number of bytes. The address and count are hexadecimal numbers. If the count is not specified, one page (100h bytes) will be display. The D command can be repeated by pressing ↵ Enter at the # prompt.

```
# D FFF7765E

FFF7765E  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF7766E  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF7767E  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF7768E  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF7769E  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF776AE  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF776BE  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF776CE  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF776DE  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF776EE  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF776FE  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF7770E  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF7771E  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF7772E  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF7773E  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF7774E  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................

# D FFF7765E 10

FFF7765E  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
```

M *address [* L *length] bytepattern*

Search memory for a *bytepattern* match, starting at location *address* and continuing until */*L *length]* is reached. If a match is found, 128 bytes (beginning with the pattern) are displayed. The M command can be repeated by pressing ↵ Enter at the # prompt.

```
# M FFF77F00 54 48 45 52

FFF77F1C  54 48 45 52 4E 45 54 5F - 49 49 00 90 00 00 00 00  THERNET_II......
FFF77F2C  00 00 00 00 00 00 90 6B - F7 FF 00 00 00 00 00 00  .......kw.......
FFF77F3C  48 61 72 64 77 61 72 65 - 44 72 69 76 65 72 4D 4C  HardwareDriverML
FFF77F4C  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF77F5C  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF77F6C  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF77F7C  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
FFF77F8C  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................

# M FFF77F5C L1F 54 48

Match not found
```

Register Manipulation

This section describes the debugger commands used on the microprocessor's general and flag registers.

R

Display the EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP, EIP, and Flag Registers.

```
# R

EAX=99999999  EBX=00005455 ECX=78787878 EDX=00060544
ESI=00000000  EDI=80868086 EBP=00000000 ESP=FFF67876
EIP=FFF56784  FLAGS=00010002
```

register = value

Change the specified *register* to the new *value*. The command is effective with EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP, and EIP.

```
# EAX=8099ACB3

Register changed
```

F *flag = value*

Change the specified *flag* to the new *value* (0 or 1). The command is effective with the CF, AF, ZF, SF, IF, TF, PF, DF, and OF flags.

```
# F PF=0

Flag changed
```

Input/Output

This section describes the debugger's I/O commands.

I*[B,W,D] port*

Input a byte, word, or double word from *port*.

```
# I 25A
Port (25A)=F8
# IB 25A
Port (25A)=F8
# IW 1B3
Port (1B3)=D3FF
```

O*[B,W,D] port = value*

Output a byte, word, or double word *value* to *port*.

```
# O 25B=7D
Output completed
# OW 18E=3C0F
Output completed
```

Miscellaneous

This section consists of descriptions of the remaining debugger commands.

G *[address(es)]*

Begin execution (Go) from current position and set temporary breakpoint *[address(es)]*.

```
# G FFF56784

Break at FFF56784 because of go breakpoint
EAX=99999999  EBX=00005455  ECX=78787878  EDX=00060544
ESI=00000000  EDI=80868086  EBP=00000000  ESP=FFF67876
EIP=FFF56784  FLAGS=00010002

FFF56784 BB30CE0500  mov  ebx, 0005CE30
```

N *symbolname value*

Define a new symbol with a value.

```
# N thissym 0F0F
```

P

Proceed over next instruction. This command is similar to the "Trace" or "Single Step" command but it will not single step loops or calls. The P command can be repeated by pressing ↵ Enter at the # prompt.

Q

Quit and return to DOS.

T or S

Trace or Single Step through the program. The T or S commands can be repeated by pressing ↵ Enter at the # prompt.

U *address [count]*

Unassemble *count* instructions from *address*. The U command can be repeated by pressing ↵ Enter at the # prompt.

```
# u FFF87885 2

FFF87885  0000     add   [eax], al
FFF87887  0000     add   [eax], al
```

V

View the screens (will step through the screens sequentially).

Z *expression*

Evaluate the expression. (calculator)

```
# z 7+8

Evaluates to:  F
```

Debug Expressions

All numbers in debug expressions are entered and shown in hex format. In addition to numbers, the following registers, flags, and operators can be used in expressions and breakpoint conditions:

Registers: `EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP, EIP`
Flags: `FLCF, FLAF, FLZF, FLSF, FLIF, FLTF, FLPF, FLDP, FLOF`

Operators and precedence:

Symbol	Description	Precedence
!	logical not	1
-	2's compliment	1
~	1's compliment	1
*	multiply	2
/	divide	2
%	mod	2
+	addition	3
-	subtraction	3
>>	bit shift right	4
<<	bit shift left	4
>	greater than	5
<	less than	5
>=	greater than or equal to	5
<=	less than or equal to	5
==	equal to	6
!=	not equal to	6
&	bitwise AND	7
^	bitwise XOR	8
|	bitwise OR	9
&&	logical AND	10
\|\| | logical OR | 11

Grouping Operators

The operators **( )**, **[ ]**, and **{ }** have a precedence of 0. These grouping operators can be nested in any combination.

(*expression*)
Causes the expression to be evaluated at a higher precedence.

[*size expression*]
Causes the expression to be evaluated at a higher precedence and then uses the value of the expression as a memory address. The bracketed expression is replaced with the byte, word, or double word at that address. "Size" is a data size specifier of the type B, W, or D.

{*size expression*}
Causes the expression to be evaluated at a higher precedence and then uses the value of the expression as a port address. The bracketed expression is replaced with the byte, word, or double word input from the port. "Size" is a data size specifier of the type B, W, or D.

Conditional Evaluation

expression1* ? *expression2* , *expression3
If *expression1* is true, then the result is the value of *expression2*; otherwise, the result is the value of *expression3*.

Symbolic Information

Symbolic information may be included in a driver file that can be used to access routines or variables by name while in the NetWare 386 debugger. To access symbolic information, the following steps must be taken:

1) Declare public all desired symbols in the driver.
2) Include the keyword *debug* in the driver's linker definition file.

Each of these symbol names (the debugger is case-sensitive) can now be used in the same way the address they represent would be used. For example, at the debug prompt it is possible to display memory beginning at the address of the label *AdapterBdStruct* by entering:

```
#d AdapterBdStruct
```

Symbols may be dynamically defined by the debugger. If it is necessary to dynamically define more than 10 symbols the server must be loaded with the **-y** option.

Note: Debugging information *must* be removed before releasing the driver. Including the *debug* keyword in the definition file will cause a message to be displayed on the console when the driver is loaded, indicating that it contains debug information.