MSDebug manual

2023 by C. Masloch. Usage of the works is permitted provided that this instrument is retained with the works, so that any entity that uses the works is notified of this instrument. DISCLAIMER: THE WORKS ARE WITHOUT WARRANTY.

This document has been compiled on 2023-07-24.

Contents

Section 1: Overview and highlights

MSDebug is a 86-DOS debugger based on the original Debug from Microsoft. It is based on the Debug sources of the 2018 free software release of MS-DOS version 2. A few features from newer Debug versions were recreated. Further, a few fixes and extensions have been added. However, two features are still missing:

- EMS related X commands
- L/W command disk sector accesses for file systems with more than 65535 sectors, or FAT32 file systems

MSDebug is intended to stay largely compatible to Debug. For a more powerful advanced debugger, consider lDebug, based on the FreeDOS Debug/X project. (Refer to section 12 for a list of lDebug features.)

1.1 Quick start for reading this manual

- The interface reference explains the basics of the debugger's interface and lists some common commands.
- The parameter reference lists the types of parameters used by the available commands.
- The command reference describes most commands in detail.
- The interrupt reference lists what interrupt hooks the debugger sets up.
- Usage conditions list the license and copyright attributions.
- The lDebug advertisement states why you should use lDebug instead.
- Source Control Revision ID gives the Mercurial hash of the manual's source revision, and links to that revision in ecm's repository.

1.2 Changes from original 2018 source release Debug

- P command support for repetition count added (recreated from later MS-DOS versions)
- ORG directive added to assembler (recreated from later MS-DOS versions, though with better error handling)
- Interrupt 1 and interrupt 3 vectors are restored (upon return into the debugger, enabling to debug most of the debugger itself like lDDebug)
- Tracing into a software interrupt can be done using T command, use the P command to proceed past interrupt instead (recreated from later MS-DOS versions)

- K command that acts like lDebug's K command (or lDebug's default handling of the N command)
- BU command like lDDebug's BU command
- Child process is allocated memory using int 21h function 48h, and initialised with function 55h
- After the attached process has terminated a child process is created again
- If a child process cannot be created due to lack of memory (272 bytes required) then the comma command ‘`,`’ can be used to retry creation
- Child process is created with the same correct CALL 5 address as used by more modern MS-DOS and FreeDOS kernels
- Data is aligned to even boundaries
- Online help screen and command help added
- Created child process is initialised with a `retn` instruction
- Child process stack is initialised properly with a zero word
- Bugfix: A program is loaded initially even when the command line tail after the program load filename is empty
- .HEX file read fixed to end on encountering a NUL byte (0), EOF byte (1Ah), or the EOF (a short read)
- Do not leak file handles after program-loading or program-saving commands
- .HEX file read aborts if it wraps around the offset
- Bugfix: Move command M will correctly move forwards or backwards based on the linear address, fixing overlapping moves in which the segments differ in the opposite way to the linear addresses

Section 2: Invoking the debugger

One switch is supported:

`/?`

Show the command help page about invoking the debugger. Refer to section 9 for a copy of that help.

Section 3: Interface Reference

3.1 Interface Output

The debugger provides a line-based text interface. The interface is written to DOS standard output.

3.2 Interface Input

The default command prompt indicates that a command may be entered. It is a dash '-'. Input is read from DOS standard input.

If the input is redirected from a file (a Debug script) it is crucial to include a Q command in the file. The lDebug sources claim that MSDebug may hang if it reaches the End Of File on standard input.

3.3 Register dumping

The R command (refer to section 6.18) without any parameters dumps the current register values. Then it disassembles a single instruction. The register dump looks like this:

```
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=42A4  ES=42A4  SS=42A4  CS=42A4  IP=0100   NV UP DI PL NZ NA PO NC
42A4:0100 C3            RET
-
```

After running the program being debugged, usually the R command is also being run. This includes a step with the T or P commands. (Section 6.20, section 6.16.) It also includes a run with the G command. (Section 6.8.)

3.4 Memory dumping

Another basic command is the D command (section 6.5). It is used to dump memory contents. For example, to dump part of a program:

```
-d
42A4:0140  8C C8 31 DB 05 59 19 50-53 CB 26 80 7F 02 00 74   .H1[.Y.PSK&.
42A4:0150  14 26 C7 47 03 00 01 26-80 7F 02 0E 74 06 26 C7   .&GG...&....
42A4:0160  47 03 03 81 CB 50 1E 9C-53 BB 25 05 EB B5 5B 06   G...KP..S;%.
42A4:0170  E9 59 73 59 9D 13 20 63-B9 64 F6 2F 3C 65 67 67   iYsY.. c9dv/
42A4:0180  E5 7B 08 6D 55 69 BD 6B-E9 6C D6 66 0A 3E 1C 6F   e{.mUi=kilVf
42A4:0190  A3 1F 31 76 37 39 D1 45-76 7C 5E 78 CE 79 4D A3   #.1v79QEv|^x
42A4:01A0  00 00 00 00 10 41 00 00-0F 00 00 60 02 00 00 00   .....A.....`
42A4:01B0  00 00 00 00 00 00 01 00-00 00 00 00 10 41 00 00   .............
-
```

3.5 Disassembly

The U command is used to disassemble one or several instructions. Example:

```
-u
5BFD:0000 8CD0          MOV     AX,SS
5BFD:0002 8CDA          MOV     DX,DS
5BFD:0004 29D0          SUB     AX,DX
5BFD:0006 31D2          XOR     DX,DX
5BFD:0008 B90400        MOV     CX,0004
5BFD:000B D1E0          SHL     AX,1
5BFD:000D D1D2          RCL     DX,1
5BFD:000F E2FA          LOOP    000B
5BFD:0011 50            PUSH    AX
5BFD:0012 01E0          ADD     AX,SP
5BFD:0014 83D200        ADC     DX,+00
5BFD:0017 83C00F        ADD     AX,+0F
5BFD:001A 83D200        ADC     DX,+00
5BFD:001D 24F0          AND     AL,F0
5BFD:001F 83FA02        CMP     DX,+02
-
```

3.6 Loading the debuggee

A program to examine can be loaded using the N or K, then L commands. If the debugger is loaded with a filename specified in its command line, it will run the K and L commands on its own.

The K command sets up some buffers internal to the debugger. These are *only* used when a new process comes to be, either because the attached process has terminated or because a program-loading L command is used.

One of those buffers specifies the pathname of the executable file to load. The pathname must include the filename extension, if any. The pathname must be relative to the current directories at the time the L command runs, or it must be absolute. The tail of the K command after the pathname is used as the command line tail for a new debuggee process.

The N command behaves a little differently in MSDebug. The initial pathname is used as the program load filename in the same way as for the K command. However, the N command will use all its parameters as the command line tail, not just the remainder after the pathname. Further, it allcapses the command line tail. Finally, the N command will always write both the debugger's internal buffers as well as the PSP assumed to be addressable using the current debuggee DS register.

The L command without any parameters attempts to load the program specified to the last N or K command into a new process. If the L command does not display any messages this indicates success.

3.7 Running the debuggee

Once a program is loaded into the debugger it can be run in several ways:

G command

Runs at full speed until a breakpoint is encountered. Temporary breakpoints can be specified to the G command. Refer to section 6.8.

T command

Traces a single instruction. Refer to section 6.20.

P command

Either runs at full speed with a breakpoint behind the current instruction, or traces a single instruction. Software interrupts, call instructions, repeated string instructions, and loop instructions are proceeded past by using a breakpoint. Refer to section 6.16.

3.8 Help

The online help can be accessed using the '?' command. Refer to section 10 for a copy of the online help.

Section 4: Debugging the debugger itself

The debugger installs its interrupt handlers only within the ‘`dexit`’ function, so as to return the control flow to this instance when it runs its debuggee code. On return into this instance, it uninstalls its handlers again. This mechanism allows to debug most of the debugger using a different debugger.

An additional command is supported, the BU command (which stands for "Break Upwards"). It will run a breakpoint within the debugger's code segment which will break into the other debugger. Its code was updated so it will break at the command dispatcher. This means if the outer debugger is an lDebug then it can be instructed to skip to the next command being dispatched by entering the command ‘`G ip`’.

Other than for the most trivial sessions it is recommended to control the outer debugger by serial I/O, separately from the I/O of the debuggable debugger.

Section 5: Parameter Reference

5.1 Number

Plain numbers are read in hexadecimal, up to 4 hexits. Plain number parameters are used by a lot of commands. Sometimes, the plain number parameter type is called ‘byte’ or ‘value’.

5.2 Address

An address parameter is calculated with a default segment. First, a plain number may be parsed. If it is followed by a colon, the first number is taken as segment, and then another number is parsed for the offset.

Instead of a segment number, the name of one of the 4 segment registers may be specified. In this case the colon is mandatory, and a number for the offset must follow it.

Otherwise, the first number is used as the offset. Offsets are 16 bits.

Address parameters are used by a lot of commands.

5.3 Range

A range parameter may have a default length, or it may be disallowed to omit a length. Parsing a range starts with parsing an address. Then, if the end of the line is not yet reached, an end for the range may be specified. The end may be a plain number, which is taken as the offset of the last byte to include in the range. The address of the last byte to include must be equal or above the address of the first byte that is included in the range. Specifying a start offset of 0 with an end offset of 0FFFFh is invalid because the maximum length is 0FFFFh.

The end may instead be specified with an ‘`L`’ keyword. In that case, the keyword is followed by a plain number. The maximum length is 0FFFFh. A length of zero is handled in a special way. For most commands parsing ranges, a length of zero indicates to operate on a full 64 KiB segment. Unlike other lengths, the zero length may cause a wrap around from offset 0FFFFh to 0000h. An exception is the U command, which treats a zero length in the same way as a 1 length.

Range parameters are used by a lot of commands.

5.4 List

A list is made up of a sequence of items. Each item is either a plain number or a quoted string. List parsing continues until the end of the line. Each plain number represents a single byte. Quoted strings represent as many bytes as there are quoted. A quoted string can be delimited by single quotes `'` or double quotes `"`. If the used delimiter quote mark occurs twice back to back while reading the quoted string, this is taken as an escape to include the delimiter mark itself

as a byte of the string. List parameters are used by the E, F, and S commands. Refer to section 6.6, section 6.7, and section 6.19.

5.5 Breakpoint

Each breakpoint is a single address, which defaults to the code segment. The breakpoint parameter type is used by the G command, refer to section 6.8.

5.6 Port

A port is a plain number for parsing purposes. The port parameter type is used by the I and O commands, refer to section 6.10 and section 6.15.

5.7 Drive

A drive is a plain number. The number zero corresponds to drive A:. The drive parameter type is used by the L and W sector commands, refer to section 6.12 and section 6.23.

5.8 Sector

A sector is a plain number, which can be equal to any 16-bit value. The sector parameter type is used by the L and W sector commands, refer to section 6.12 and section 6.23.

5.9 Register

A register specifies an internal variable of the debugger. These are the debuggee's registers as stored by the debugger in its data segment. One form of the R command uses a register parameter. This allows reading and writing the register values. Refer to section 6.18.

Section 6: Command Reference

6.1 ? command

```
Online help  ?
```

The question mark command (?) lists the online help screen. The full help page is listed in section 10.

6.2 A command - Assemble

```
assemble     A [address]
```

Starts assembly at the indicated address (which defaults to CS segment), or if no address is specified, at the "asmadd".

Assembly mode has its own prompt. Entering an empty line terminates assembly mode. Comments can be given with a prefixed semicolon.

6.3 BU command - Break Upwards

This command causes the debugger to execute an int3 instruction in its own code segment. This breaks to the next debugger that was installed prior to MSDebug.

6.4 C command - Compare memory

```
compare      C range address
```

Given a range, the address of which defaults to DS, and another address that also defaults to DS, this command compares strings of bytes, and lists the bytes that differ.

6.5 D command - Dump memory

```
dump         D [range]
```

Given a range, the address of which defaults to DS, this command dumps memory in hexadecimal and as ASCII characters. The default length if none is specified defaults to 128 bytes.

In the text dump, control characters are replaced by dots, while bytes with their high bit set are treated as if the high bit was clear.

If no range is specified, the D command continues dumping at "defdump", which is updated by each D command to point after the last shown byte.

6.6 E command - Enter memory

```
enter        E address [list]
```

The E command is used to enter values into memory. If the list is specified, its contents are written to the address specified. Otherwise, the interactive enter mode starts at the address specified.

In the interactive enter mode, the segmented address is displayed, and then the current byte value (2 hexadecimal digits) found at that address yet. Following the value a dot is displayed. For example:

```
-e 100
08BD:0100  C3.
```

At this point the debugger accepts several different inputs:

- One or two hexadecimal digits: To enter a new value to be written at this address
- A blank: To write the new value (if any) and proceed to the next byte
- A minus: To write the new value (if any) and proceed to the prior byte
- Carriage Return: To write the new value (if any) and quit interactive enter mode
- Backspace: To delete the most recently entered digit of a candidate new value
- All other inputs are ignored

After entering a blank, the debugger will either display the next byte's current value in the same line or start a new line with the current segmented address and then the current byte value. A new line is started if the current offset is divisible by 8. For example, after entering 8 blanks:

```
-e 100
08BD:0100  C3.     CC.     CC.     CC.     CC.     CC.     CC.     CC.
08BD:0108  CC.
```

After entering a minus, the minus is displayed on the current line and then (always) a new line is started to display the new segmented address (with its offset decremented). For example, entering a new value (‘A0’), then a blank, then a minus, and then another new value (‘A1’), then a CR:

```
-e 100
08BD:0100  C3.A0   CC.-
08BD:0100  A0.A1
-
```

6.7 F command - Fill memory

```
fill         F range list
```

The F command fills memory with a byte pattern. The first parameter is the range to fill. The next parameter is a list, which provides the pattern with which to fill. The pattern is repeated so as to fill the destination.

6.8 G command - Go

```
go              G [=address] [addresses]
```

The G command runs the debuggee. It can be given a start address (the segment of which defaults to CS), prefixed by an equals sign, in which case CS:IP is set to that start address upon running.

The G command allows specifying breakpoints, which are segmented addresses. By default, 10 G breakpoints are supported.

6.9 H command - Hexadecimal add/subtract values

```
hex             H value1 value2
```

The H command performs calculation and displays the results. The first result is that which is calculated by adding the two numbers. The second result is calculated by subtracting the second number from the first number. The results are written as unsigned 16-bit numbers in hexadecimal, 4 hexits per number.

6.10 I command - Input from port

```
input           I port
```

The I command inputs from an x86 port. The port can be any number between 0 and FFFFh. I inputs a byte from the specified port.

6.11 L command - Load Program

```
load            L [address]
```

6.12 L command - Load Sectors

```
load            L [address] [drive] [firstsector] [number]
```

6.13 M command - Move memory

```
move            M range address
```

6.14 N command and K command - Set program Name

```
name            N [pathname] [arglist]
set command     K [pathname [arglist]]
```

These commands set up the filename and parameters to use when setting up a new process using the L (Load program) command. If the filename ends in `.COM` or `.EXE` it will be loaded as a DOS program using the interrupt 21h service 4B01h. If the filename ends in `.HEX` the debugger will parse it as an Intel hex file. Otherwise the file is loaded as a flat binary by the debugger itself. In any case, the PSP of the process created by the L command will receive the command line tail, which for K starts after the filename.

Unlike the N command, for the K command the executable filename is not included in the command line tail, and an existing process won't be modified by the K command. It only sets the filename and tail for L to use.

The N command writes to memory between DS:005Ch and DS:0100h. It stores the command

line tail in allcaps.

The K command was added to MSDebug to provide the same experience as lDebug's K command, which matches lDebug's default N command. The K command behaves in a similar way to how MSDebug handles a program load pathname and command line tail specified on the MSDebug command line initially.

6.15 O command - Output to port

```
output      O port byte
```

The O command outputs to an x86 port. The port can be any number between 0 and FFFFh. O outputs a byte to the specified port. The value to write is specified by the second number.

6.16 P command - Proceed

```
proceed     P [=address] [number]
```

The P command causes debuggee to run a proceed step. This is the same as tracing (T command) for most instructions, but behaves differently for '`int`', '`call`', '`loop`', and repeated string instructions. For these, a proceed breakpoint is written behind the instruction (similarly to how the G command writes breakpoints), and the debuggee is run without the Trace Flag set.

Like for the G command, a start address can be given to P prefixed by an equals sign. Next, a count may be specified, which causes the command to execute as many P steps as the count indicates.

6.17 Q command - Quit

```
quit        Q
```

6.18 R command - Display and set Register values

```
register    R [register]
```

The R command without any register specified dumps the current registers, and disassembles the instruction at the current CS:IP location.

R with a register displays the current value of the specified variable. It then displays a prompt, allowing the user to enter a new value for that variable. Entering an empty line returns to the default debugger command line.

6.19 S command - Search memory

```
search      S range list
```

The S command searches memory for a byte string. The range specifies the search space. The search string is specified as a list of byte values. The display of search results consists of the result's segmented address.

6.20 T command - Trace

```
trace       T [=address] [number]
```

The T command is similar to the P command. However, T traces all instructions.

6.21 U command - Disassemble

```
unassemble   U [range]
```

Given a range, the address of which defaults to CS, this command disassembles instructions from memory. The default length if none is specified defaults to 32 bytes. All instructions that are contained within or start within the specified range are disassembled.

If no range is specified, the U command continues disassembling at "disadd", which is updated by each U command to point after the last disassembled byte. The default length is the same as for if a range without a length is specified.

6.22 W command - Write Program

```
write        W [address]
```

6.23 W command - Write Sectors

```
write        W [address] [drive] [firstsector] [number]
```

6.24 Comma command

If the debuggee terminated but did not release enough memory to create an empty process (a memory block of 272 bytes), then the debugger will emit an error message. Entering a command consisting only of a comma, followed directly by a Carriage Return, will attempt to create a process again. This command always displays a message stating what it did.

Section 7: Variable Reference

Most 16-bit 8086 debuggee registers can be accessed using the R command. These are:

- `ax`, `cx`, `dx`, `bx`, `sp`, `bp`, `si`, `di`
- `es`, `cs`, `ss`, `ds`
- `ip`

Section 8: Interrupt Reference

- Interrupt 1 - Trace
- Interrupt 3 - Breakpoint

These interrupts are hooked by the debugger.

These interrupts are hooked within the `dexit` function and unhooked before the `dexit` function returns.

Unhooking is always done by simply updating the IVT entries with whatever handlers are stored as the prior vectors.

Section 9: Command help

```
MSDebug release 0 by ecm

Runs Debug, a program testing and editing tool.

DEBUG [[drive:][path]filename [testfile-parameters]]

  [drive:][path]filename  Specifies the file you want to test.
  testfile-parameters     Specifies command-line information required by
                          the file you want to test.

After Debug starts, type ? to display a list of debugging commands.
```

Section 10: Online help

```
MSDebug release 0 by ecm help screen
assemble     A [address]
compare      C range address
dump         D [range]
enter        E address [list]
fill         F range list
go           G [=address] [addresses]
hex          H value1 value2
input        I port
load         L [address] [drive] [firstsector] [number]
move         M range address
name         N [pathname] [arglist]
set command  K [pathname [arglist]]
output       O port byte
proceed      P [=address] [number]
quit         Q
register     R [register]
search       S range list
trace        T [=address] [number]
unassemble   U [range]
write        W [address] [drive] [firstsector] [number]
```

Section 11: Usage conditions and attributions


```
Copyright (C) 1983 Microsoft Corp.
Modifications copyright 2018 John Elliott
          and copyright 2022 S. V. Nickolas.
```

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the Software), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED AS IS, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

MS-DOS is a Registered Trademark of Microsoft Corp.

Section 12: lDebug advertisement

This section lists some benefits of lDebug, the FreeDOS Debug/X fork.

- Expression evaluator can be used whereever a number is to be parsed
- D, U, T, TP, P, G commands can be repeated with autorepeat
- Permanent breakpoints (B commands)
- G command can re-use prior breakpoints list with G AGAIN command (or autorepeat)
- Several variables beyond the 16-bit registers to control the debugger or store calculation results
- Paging to allow reading longer outputs
- 686 level assembler and disassembler
- DPMI build lDebugX for debugging DPMI clients, supporting 32-bit offsets in segments
- InDOS mode to switch to ROM-BI0S video and keyboard I/O rather than DOS's standard output and input
- Line editor and line history (if not using the DOS line input function)
- Can be boot loaded for debugging Real/Virtual 86 Mode kernels
- Can be installed as a device driver in CONFIG.SYS
- Script for lDebug file reading using the Y command
- Serial I/O mode
- Conditional tracing for the T, TP, and P commands using a condition after a WHILE keyword
- Buffered tracing for T, TP, and P using a SILENT keyword
- T defaults to proceeding past software interrupt calls, can be modified using Trace Mode (TM command)
- TP command which proceeds past repeated string instructions
- DM command to list MCBs
- DI command to dump interrupt handler chains
- DW and DD commands to dump data in words or dwords

- 32-bit numeric handling
- F and S commands can use a memory source instead of a list, using a RANGE keyword
- H command can display an expression result in hexadeximal, decimal, or one arbitrary base of choice
- I and O commands have IW/OW and ID/OD variants for word and doubleword port I/O
- R command can be switched to display 32-bit and 386 registers
- Machine type can be set to make the assembler and disassembler display when the machine does not support an instruction
- QA command can be used to try to terminate an attached process
- S command can search backwards using the REVERSE keyword
- Provides a number of online help pages
- RN command to dump 8087 registers
- RM command to dump MMX registers, and variables to read and write them
- RE command buffer to run commands from T, TP, P, or G dump calls
- RC command buffer to run commands at startup or group several commands later on
- Numeric inputs can be specified with a hash sign '#' modifier to enter in arbitrary numeric bases
- Access variables and VALUE IN constructs to detect memory accesses before they are actually carried out (requring to trace instructions)
- TSR and ATTACH commands to detach from or attach to a process
- IF command to conditionally run another command
- Timer and AMIS interrupt hooks (optional)

Furthermore, lDebug can be built using the free software Netwide Assembler, rather than relying on a binary-only Microsoft Macro Assembler. (The assembler executable shipped with MSDebug is technically free, but it does not have sources.)

However, there are some disadvantages to lDebug as well:

- Memory use and executable size can easily reach as much as ten times that of MSDebug
- Less compatibility to original MS-DOS Debug
- Performance may be worse
- May require some MS-DOS version 3 or version 5 features

Source Control Revision ID

hg 42b1286b1b63, from commit on at 2023-07-24 20:30:49 +0200

If this is in ecm's repository, you can find it at https://hg.pushbx.org/ecm/msdebug/rev/42b1286b1b63