DD-DOS

A Small DOS Virtual Machine

DD-DOS Technical Manual

Revision 2018.0, 2018-04-28

Revision History

Revision	DD-DOS V.	Date	Description
v0.0.0	v0.0.0	2017-01-16	Internal Initial Release
v0.1.0	v0.0.0	2017-01-17	Additions
v0.2.0	v0.0.0	2017-01-23	Additions, improvements, and corrections.
v0.2.1	v0.0.0	2017-01-23	Internal Shell and fixes.
v0.3.0	v0.0.0	2017-02-01	Added Sections, debugging additions, file formats.
v0.4.0	v0.0.0	2017-03-01	Additions, small fixes.
v0.5.0	v0.0.0	2017-04-22	First public release: A few additions, a lot of corrections.
v0.5.1	v0.0.0	2017-04-22	Fixed table of content and figures.
v0.6.0	v0.0.0	2017-04-29	Added and re-organized some major sections: X86 Encodings, Micro-processors, Systems, Operating systems, Glossary, and Index sections were added. Modified all legends to label tables as such instead of figures. Many English-related and typographic issues resolved.
v0.6.1	v0.0.0	2017-04-29	Added REG field encoding table, removed one confusion.
v0.6.2	v0.0.0	2017-04-29	Added 32-bit ModR/M table. Fixed a few things including possible confusion in §4.
v0.6.3	v0.0.0	2017-12-28	Modifications (logger, cli, disclaimer, document styles a bit) Changed style
v0.6.4	v0.0.0-0	2018-02-04	Corrections for clarity
v0.6.5	v0.0.0-0	2018-03-26	Changed from Reference Manual to Technical Manual Changed copyright information
v0.7.0	v0.0.0-0	2018-04-08	Added x86 instruction format table Adjusted fonts Redone page numbering Revisited every articles, fixed typos and sentence structure
v0.7.1	v0.0.0-0	2018-04-13	Fixed typos Modified footer style Revisited Glossary
2018.0	v0.0.0-0	2018-04-28	Whole document corrections and adjustments, inc. footer Add x86 flags

Table of Content

Table of Tables

Preface

About This Document

This document is about the internal technical details of the DD-DOS (https://github.com/dd86k/dd-dos) project for developers, engineers, enthusiasts, and the curious mind. It was initially written for myself as memory aid.

This document is written by dd86k (https://github.com/dd86k), it is a collection of information gathered through out on the web and various documentation. Any comments, additions, suggestions, and fixes can be sent via email or as an Issue on Github with the label “Technical Manual”.

Happy reading!

Conventions

This document uses various conventions and uses X as the placeholder, typically a number.

Xh	Hexadecimal-formatted number, usually grouped by two bytes (e.g. FF_FFFFh).
Xb	Binary-formatted number, the b suffix may be optional.
`Mono-spaced font`	Code, or command example.
X:X	Bit range, both the minimal and maximum numbers are inclusive.

Table 1: Conventions

Byte-order

All number references are little-endian, therefore a number such as AABB_CCDDh will have DD as the first, most meaningful byte.

Trademarks

MS-DOS and Windows are trademarked Microsoft products.
All referenced Intel products, like the Intel 8086, are trademarked Intel products.

Disclaimer

The information contained within this document are solely advisory, should not be substituted for professional advice, and may change at any given time without any warning.

1 Introduction

The Microsoft Disk Operating System, mostly known as MS-DOS and derived from 86-DOS, was the most used DOS-based operating system from its time, making it a key product for Microsoft to grow as a rather large software company during the time.

Ever since the release of 64-bit Windows operating systems, the 16-bit emulation layer, NTVDM, has been stripped out of the system due to x86-64 based processors not supporting the Virtual 8086 mode while running in LONG mode (64-bit).

1.1 Goal

DD-DOS is an open-source project aiming and hoping to be a light and simple DOS-compatible layer, running alongside the very operating system's terminal via emulation on many platforms and may eventually replace NTVDM, the WindowsNT Virtual DOS Machine.

In the near future, dd-dos will support both the Intel 8086 and the Intel i486 as emulated micro-processors, with added mouse support.

In the far future, a graphical environment and audio system could be implemented, spawning as a separate window if the DOS program requests a visual graphical mode.

2 Initialization

2.1 Command Line Interface

dd-dos supports short POSIX condensed notation for certain parameters.

Without a FILE argument, dd-dos starts into its virtual shell and uses vDOS, its internal virtual operating system.

Syntax:

```
dd-dos [OPTIONS] [FILE]
dd-dos {-v|--version|-h|--help}
```

Options:

`--version`

Displays the version of the project, and exits.

`-h, --help`

Displays a help screen and exits.

`-V`

Enable verbose mode. Not recommended for TUI applications.
Debug builds automatically sets verbose mode. It is possible to turn it off again via -V.

`-P`

Disables thread sleeping.

`-N`

Disables start-up banner logo and start-up messages.

3 Settings

User settings are saved in a file named `"settings.sdl"` under the user profile (under `~/.dd-dos` or under the current directory) using the YAML 1.2 specifications (http://yaml.org/) formated in UTF-8. By default, DD-DOS will not create a settings file if missing, and will assume default settings. If fields are missing, DD-DOS will assume default values for those parameters.

Settings are currently not implemented.

Parameter name	Type	Default value	Description
row	Number	24	Set console height
column	Number	80	Set console width
app	String	null	Set starting application
date	Number	Current date	Set date
time	Number	Current time	Set time
cwd	String	.	Set working directory
videomode	Number	-1	Specifies a video mode A value of -1 assumes defaults

Table 2: DD-DOS Settings

3.1 Console size

DD-DOS requires at least 80x24 characters to operate normally. If the console window is too small, it will be resized if possible, otherwise it will terminate with an error message.

3.2 Date and time

The date and time formats are 32-bit hexadecimal numbers expressed in little-endian.

For time, all fields are optional except for hours. For example, 0225_0A06h converts to 6:10:25.02, and 1007h converts to 7:16:00.00.

Offset	Size	Description and Register
0	1	Hours (CH)
1	1	Minutes (CL, Optional)
2	1	Seconds (DH, Optional)
3	1	Milliseconds (DL, Optional)

Table 3: Time Setting Structure

For date, all fields are required. For example, a number 07CB0203 would convert to 1995/Feb/03.

Offset	Size	Description and Register
0	2	Year (CX)
2	1	Month (DH)
3	1	Day (DL)

Table 4: Date Setting Structure

4 x86 Instruction Format

This section attempts to address some of the ways the x86 machine code is encoded.

A typical instruction is encoded with the direction bit (D) and the word bit (W).

OPERATION CODE							
						D	W

Table 5: x86 Opcode Encoding

The D bit is set whenever the destination is specified in the REG (see: ModR/M Byte) field. Otherwise, the register specified in the REG field is used as the source.

The W bit is set whenever the instruction operates on WORD data (2 bytes). Otherwise, BYTE data.

4.1 ModR/M Byte

The Mode Register/Modifier byte is an instruction modifier. It can set a memory mode, destination or source register, etc.

The ModR/M byte contains three fields: MOD, REG, and R/M.

ModR/M Byte							
MOD		**REG**			**R/M**		

Table 6: ModR/M Byte Structure

MOD	Description
00	Memory Mode, no displacement follows (except when R/M is 110)
01	Memory Mode, 8-bit displacement follows
10	Memory Mode, 16-bit displacement follows
11	Register Mode (no displacement)

REG	Register	
	Byte (W = 0)	Word (W = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP

Table 7: MOD Field Encoding

110	DH	SI

111	BH	DI

Table 8: REG Field Encoding

The effective address calculation depends from the MOD and RM fields. The REG field determines the register source or destination, depending on the destination (D) bit. Do not confuse it with the immediate value with a D prefix.

Effective Address Calculation					
R/M	**MOD = 00**	**MOD = 01**	**MOD = 10**	**MOD = 11**	**MOD = 11 + W**
000	[SI + BX]	[SI + BX] + D8	[SI + BX] + D16	AL	AX
001	[DI + BX]	[DI + BX] + D8	[DI + BX] + D16	CL	CX
010	[SI + BP]	[SI + BP] + D8	[SI + BP] + D16	DL	DX
011	[DI + BP]	[DI + BP] + D8	[DI + BP] + D16	BL	BX
100	[SI]	[SI] + D8	[SI] + D16	AH	SP
101	[DI]	[DI] + D8	[DI] + D16	CH	BP
110	DIRECT ADDRESS[1]	[BP] + D8	[BP] + D16	DH	SI
111	[BX]	[BX] + D8	[BX] + D16	BH	DI

Table 9: ModR/M Effective Address Calculation (16-bit)

The ModR/M byte's effect differ on the EXTENDED mode (i386's 32-bit mode).

Effective Address Calculation					
R/M	**MOD = 00**	**MOD = 01**	**MOD = 10**	**MOD = 11**	**MOD = 11 + W**
000	[EAX]	[EAX + D8]	[EAX + D32]	AL/EAX[2]	AX
001	[ECX]	[ECX + D8]	[ECX + D32]	CL/ECX	CX
010	[EDX]	[EDX + D8]	[EDX + D32]	DL/EDX	DX
011	[EBX]	[EBX + D8]	[EBX + D32]	BL/EBX	BX
100	SIB Mode	SIB + D8	SIB + D32	AH/ESP	SP
101	D32	[EBP + D8]	[EBP + D32]	CH/EBP	BP
110	[ESI]	[ESI + D8]	[ESI + D32]	DH/ESI	SI
111	[EDI]	[EDI + D8]	[EDI + D32]	BH/EDI	DI

Table 10: ModR/M Effective Address Calculation (32-bit)

4.2 SIB Byte

The SIB byte (Scaled Index Base), first used in the Intel i386, is used for scaled indexed addressing.

SIB Byte							
Scale		**Index**			**Base**		

Table 11: SIB Byte Structure

The SIB byte is similar to the ModR/M byte, but instead of MOD field, the SIB byte has a Scale field.

1 It is the immediate WORD that does the displacement, sign-extended.
2 The size bit in the opcode specifies 8 or 32-bit register size. To select a 16-bit register requires a prefix byte.

SIB Byte					
Scale		Index		Base	
Value	**Index * Value**	**Index**	**Register**	**Base**	**Register**
00	Index * 1	**000**	EAX	**000**	EAX
01	Index * 2	**001**	ECX	**001**	ECX
10	Index * 4	**010**	EDX	**010**	EDX
11	Index * 8	**011**	EBX	**011**	EBX
		100	ILLEGAL	**100**	ESP
		101	EBP	**101**	See note[1]
		110	ESI	**110**	ESI
		111	EDI	**111**	EDI

Table 12: SIB Byte Fields

The SIB byte heavily depends on the ModR/M byte for its effective address calculations. Note that the following table uses the Base field (B) and the Scale field (S). The immediate Displacement (D) follows the SIB byte.

Effective Address Calculation				
Index	**MOD = 00**	**MOD = 00 (B=101)**	**MOD = 01**	**MOD = 10**
000	[B32 + EAX*S]	[EAX*S + D32]	[B8 + EAX*S + D32]	[B32 + EAX*S + D32]
001	[B32 + ECX*S]	[ECX*S + D32]	[B8 + ECX*S + D32]	[B32 + ECX*S + D32]
010	[B32 + EDX*S]	[EDX*S + D32]	[B8 + EDX*S + D32]	[B32 + EDX*S + D32]
011	[B32 + EBX*S]	[EBX*S + D32]	[B8 + EBX*S + D32]	[B32 + EBX*S + D32]
100	[B32]	[D32]	[B32 + D8]	[B32 + D32]
101	[B32 + EBP*S]	[EBP*S + D32]	[B8 + EBP*S + D32]	[B32 + EBP*S + D32]
110	[B32 + ESI*S]	[ESI*S + D32]	[B8 + ESI*S + D32]	[B32 + ESI*S + D32]
111	[B32 + EDI*S]	[EDI*S + D32]	[B8 + EDI*S + D32]	[B32 + EDI*S + D32]

Table 13: SIB Byte Effective Address Calculation

1 Displacement only if MOD is cleared, otherwise EBP if MOD is set to 01 or 10.

5 Micro-processor Emulation

The only supported micro-processor at the current time is the Intel 8086.

5.1 Intel 8086

The Intel 8086 is a 16-bit micro-processor released in 1978. It was a successor of the Intel 8085 and the Intel 8080 microprocessors.

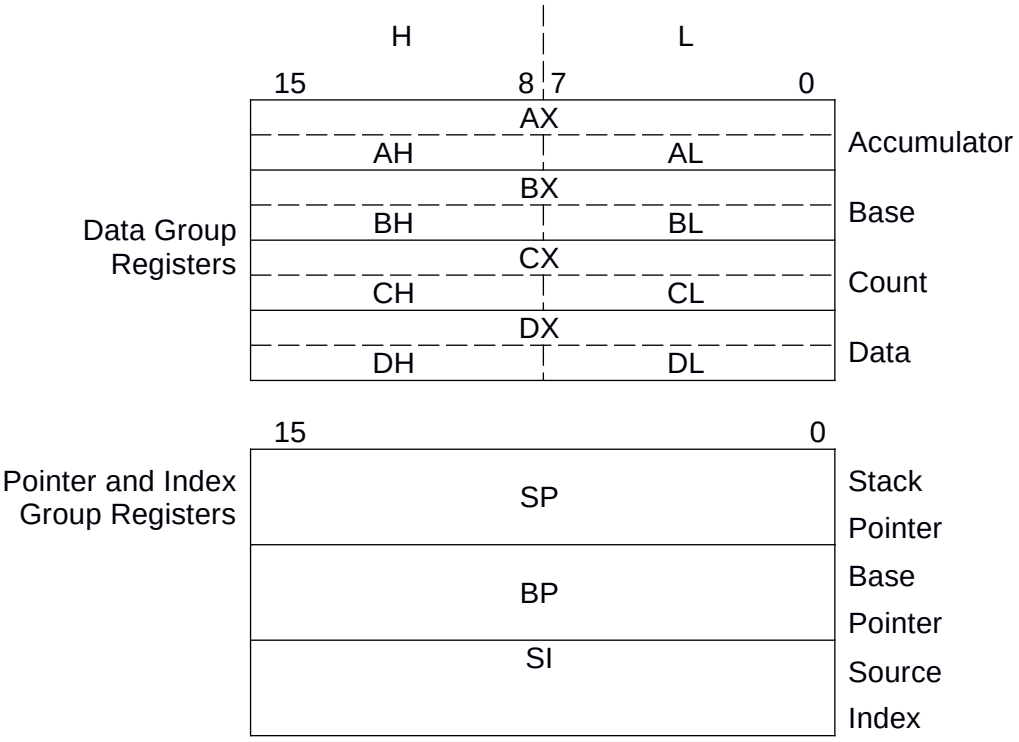
The external address bus is 20-bit wide, providing 1 MB (10_0000h) of physical address space. A linear address space is limited to 64 KB due to the 16-bit addressing and index registers. A new segmented memory model is introduced, making memory accesses into higher memory easier, which provices more than the conventional 640 KB via the segments registers.

Models are known to run between 5 and 10 MHz (2 hecto-nanoseconds and 1 hecto-nanosecond per cycle respectively plus an average of 40 nanoseconds of latency for an instruction).

5.1.1 REGISTERS

The Intel 8086 is equipped with 14 16-bit registers:

- 4 general 16-bit registers: AX, BX, CX, DX;
- 4 Index registers: SI, DI, BP, SP;
- 4 segment registers: CS, DS, ES, and SS;
- 1 Program Counter: IP; And one Flag register.



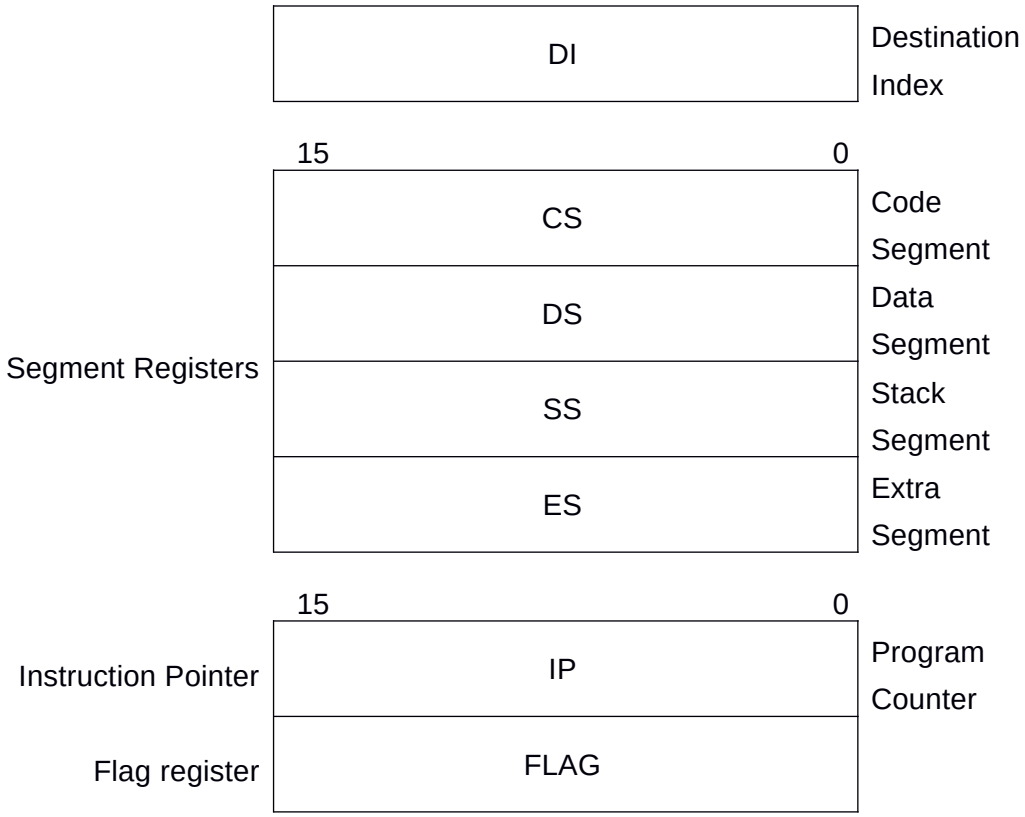


Table 14: Intel 8086 Registers

5.1.2 FLAGS

The Intel 8086 comes equipped with 9 flags. Every flag has their own position in the FLAG register.

				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Table 15: Intel 8086 Flags

The following sections describe the flags starting at the furthest bit at the right (at bit 0).

5.1.2.1 CARRY FLAG

At bit 0, when set, the CF flag indicates the high-order bit is set. That is, whenever the bit carries pat over to the destination operand; cleared otherwise.

For example, the decimal number 256 is represented as 0b1_0000_0000 in binary, thus the bit got carried to a new nibble, since a BYTE number can only hold up to 8 bits of information.

5.1.2.2 PARITY FLAG

At bit 2, when set, the PF flag indicates the low-order bits has an even number of set bits.

5.1.2.2 AUXILIARY FLAG

At bit 4, the AF flag is set if the low-order overflowed. It is primarely used with BCD arithmetics. Similar to the Carry Flag.

5.1.2.3 ZERO FLAG

At bit 6, when set, the ZF flag indicates the result is zero.

5.1.2.4 SIGN FLAG

At bit 7, when set, the SF flag indicates the sign bit, at the most left, used for signed decimal arithmetic, is set.

5.1.2.5 TRAP FLAG

At bit 8, when set, the TF flag indicates single-stepping for debugging is enabled. This flag is cleared when processing an interrupt.

5.1.2.6 INTERRUPT FLAG

At bit 9, the IF flag is set if the processor responds to maskable interrupt requests. This flag is cleared when processing an interrupt.

5.1.2.7 DIRECTION FLAG

At bit 10, the DF flag controls the flow of string instructions.

5.1.2.8 OVERFLOW FLAG

At bit 11, the OF flag is set if the result overflowed either positively or negatively.

5.1.3 RESERVED AND DEDICATED MEMORY LOCATIONS

The Intel 8086 reserves some memory for I/O mapping and functions, leaving 1,048,432 bytes (1023 KB, FFF70h) of memory and 65,528 (FFF8) I/O ports for user-applications.

Memory Locations		I/O Locations	
Reserved	F_FFFFh F_FFFCh	Open	FFFFh
Dedicated	F_FFFBh F_FFF0h		
Open	F_FFEFh 80h		100h
Reserved	7Fh 14h	Reserved	FFh F8h
Dedicated	 0	Open	 0

Table 16: Intel 8086 Dedicated/Reserved Memory and I/O Ports

5.2 Intel i486 DX

In 1982, the Intel 80286 introduced the protected mode and unsigned arithmetic operations.

In 1986, the Intel 80386, mostly known as the i386 or 386 DX, introduced the 32-bit extension of the 80286 architecture alongside the 32-bit registers.

In 1989, the Intel 80486, mostly known as the i486 DX, introduced the Floating-Point Unit, the level 1 cache (L1), and the CPUID instruction.

Intel i486 DX emulation support is planned.

6 System Emulation

6.1 IBM PC XT

Equipped with an Intel 8088, a variant of the Intel 8086 with an 8-bit communication bus, the original IBM PC was a success back in 1981, making it the bare-bone for future IBM PC compatible computers. It included PC-DOS, a variant of MS-DOS, with a memory bank that ranged from 16 KB to 256 KB of RAM.

The IBM PC XT variant, released in 1983, included a hard-drive, and memory options ranging from 128 KB to 640 KB.

This section refers to the IBM 5160 (IBM PC XT).

6.1.1 MEMORY MAP

The following memory map corresponds to the IBM 5160.

Address		Function
Decimal	**Hexadecimal**	
1024 KB	100_0000h F_0000h	F_E000 – BIOS location BIOS and BASIC 64 KB Base System ROM
768 KB	E_FFFFh C_0000h	192 KB ROM Expansion and Control C_8000h – Fixed Disk Control
640 KB	B_FFFFh A_0000h	128 KB of reserved memory B_8000h – Color graphics display B_0000h – Monochrome display
256 KB	9_FFFFh 4_0000h	384 KB of memory expansion
0	3_FFFFh 0	256 KB of Read/Write on-board memory

Table 17: IBM PC XT Memory Map

6.1.2 I/O PORT ACCESS

The IBM 5160 defines the following I/O port ranges which are accessible via the IN and OUT instructions.

Range	Description
000h-00Fh	DMA Chip (8237A-5)
020h-021h	Interrupt 8259A
040h-043h	Timer 8253-5
060h-063h	PPI 8255A-5
080h-083h	DMA Page Registers
0A0h[1]	Non-Maskable Interrupt Mask Register
0C0h	Reserved

0E0h	Reserved
200h-20Fh	Game Control
210h-217h	Expansion Unit
220h-24Fh	Reserved
278h-27Fh	Reserved
2F0h-2F7h	Reserved
2F8h-2FFh	Asynchronous Communications (Secondary)
300h-31Fh	Prototype Card
320h-32Fh	Fixed Disk
378h-37Fh	Printer
380h-38Ch[1]	Synchronous Data Link Control (SDLC) Communications
380h-389h	Binary Synchronous Communications (Secondary)
3A0h-3A9h	Binary Synchronous Communications (Primary)
3B0h-3BFh	IBM Monochrome Display/Printer
3C0h-3CFh	Reserved
3D0h-3DFh	Color/Graphics
3E0h-3E7h	Reserved
3F0h-3F7h	Diskette
3F8h-3FFh	Asynchronous Communications (Primary)

Table 18: IBM PC XT I/O Address Map

1 At power-on, the NMI into the 8088 is masked off.
To set the mask on: Send 80h. To unset: Send 00h.

1 SDLC Communications and Secondary Binary Synchronous Communications cannot be used together because their addresses overlap.

7 Operating System Emulation

7.1 MS-DOS

DD-DOS aims to emulate MS-DOS 5.0.

7.1.1 MEMORY MAP

MS-DOS follows this memory which DD-DOS SHOULD follow as well.

Command transient	High memory
Program memory	
Command resident	
MS-DOS	
I/O system	
Interrupt vectors	400h 0h - Low memory

Table 19: MS-DOS Memory Map

The `MEM /DEBUG` command SHOULD provide more information about memory usage.

8 Executable Formats

DD-DOS currently supports COM and MZ EXE executables.

File format	Details
COM	Mostly implemented; Needs tuning
MZ	Needs work
NE	Under consideration
LE	Unsupported; Not planned
LX	Unsupported; Not planned

Table 20: DD-DOS Executable Support

8.1 Program Segment Prefix

The Program Segment Prefix (PSP) is a 256-byte process table (in memory) under MS-DOS systems for the COM and MZ executables. It contains information about the running application. The PSP will be cleared alongside the application when terminated, unless INT 27h or INT 21h AH=31h is specified.

The PSP table is loaded into memory before the main application (in lower memory).

Offset	Size	Description
00h (0)	2	CP/M Exit (`INT 20h`)
02h (2)	2	First segment location
04h (4)	1	Reversed
05h (5)	5	Far call to CP/M combability mode within DOS
0Ah (10)	4	Previous programs terminate address (`INT 22h`)
0Eh (14)	4	Previous programs break address (`INT 23h`)
12h (18)	4	Previous programs critical address (`INT 24h`)
16h (22)	2	Parent's PSP segment (usually `COMMAND.COM`, internal)
18h (24)	20	Job File Table (used for file redirection, internal)
2Ch (44)	2	Environment segment
2Eh (46)	4	Entry to call `INT 21h` (SS:SP) (internal)
32h (50)	2	JFT size (internal)
34h (52)	4	Pointer to the JFT (internal)
38h (56)	4	Pointer to the previous PSP (only used by SHARE in DOS 3.3 and later)
3Ch (60)	4	Reserved
40h (64)	2	DOS version
42h (66)	14	Reserved
50h (80)	3	DOS far call (`INT 21h` and `RETF`)
53h (83)	2	Reserved
55h (85)	7	Reserved (Or can be used for extending FCB 1)
5Ch (92)	16	Unopened Standard FCB 1
6Ch (108)	20	Unopened Standard FCB 2 (overwritten if FCB 1 is opened)
80h (128)	1	Number of bytes on the command-line
81h (129)	127	Command-line (terminates with 0Dh)

Table 21: PSP Table Structure

8.2 COM Files

COM files are extremely simple raw binary files and have no header data. They cannot exceed a linear memory segment (64KB – PSP, FF00h).

To load a COM file (in MS-DOS), the DS and ES registers must be pointing at the start of the file with an origin (location counter) of 100h (PSP size), then jump to the start with an offset of 100h (CS:0100).

8.3 MZ Files

Introduced in MS-DOS 2.0, MZ files are relocatable executable files running under real-address mode.

Offset	Size	Field	Description
0	2	e_magic	“MZ” (5A4Dh) signature.
2	2	e_cblp	Number of bytes on the last page of the file.
4	2	e_cp	Number of pages in the file.
6	2	e_crlc	Number of entries in the relocation table.
8	2	e_cparhdr	Number of paragraphs[1] in the header.
10	2	e_minalloc	Number of paragraphs required by the program excluding the PSP and the program image.
12	2	e_maxalloc	Number of paragraphs requested by the program.
14	2	e_ss	Relocatable segment address for the Stack Segment (SS) register.
16	2	e_sp	Initial Stack Pointer (SP) value.
18	2	e_csum	The sum of all other fields should be zero.
20	2	e_ip	Initial Instruction Pointer (IP) value.
22	2	e_cs	Relocatable segment address for the Code Segment (CS).
24	2	e_lfarlc	Absolute offset to the relocation table.
26	2	e_ovno	Overlay management value. If zero, this is the main executable.
28	32	e_res	Extra information for the main program’s overlay management.
60	4	e_lfanew	Location of the new header, if present.

Table 22: MZ Header Structure

If the e_minalloc and e_maxalloc fields are cleared, the program is loaded in high memory. Otherwise, the program is loaded in low memory, above the PSP structure.

To load a MZ file, the code section is copied from the executable to memory. The relocation table is read and far pointers are adjusted in memory. The registers are set up with the following:

- AL and AH for the drive letter status;
- DS:ES that points to the PSP segment;
- SS:SP that points to the stack pointer.

1 A paragraph is 16 bytes in size.

The data after the header lies the relocation table. To get the position of the relocation within the file, it requires computing the physical address from the segment-offset pair. Note that the raw binary code starts on a paragraph boundary within the executable file. All segments are relative to the start of the executable in memory, and this value must be added to every segment if relocation is done manually.

Then the processor jumps to CS:IP (usually CS:0100).

8.4 NE Files

The New Executable format was introduced in the Multi-tasking MS-DOS 4.0 and Windows 1.01. It keeps the MZ stub. The header starts where `e_lfanew` points to, and complies with the following structure:

Offset	Size	Field	Description
0	2	ne_magic	“NE” (454Eh) signature.
2	1	ne_ver	Version number.
3	1	ne_rev	Revision number.
4	2	ne_enttab	Offset of Entry Table.
6	2	ne_cbenttab	Number of bytes in Entry Table.
8	4	ne_crc	CRC32 checksum of the whole file.
12	2	ne_flags	Flag word.
14	2	ne_autodata	Automatic data segment number.
16	2	ne_heap	Initial heap allocation.
18	2	ne_stack	Initial stack allocation.
20	4	ne_csip	Initial CS:IP setting.
24	4	ne_sssp	Initial SS:SP setting.
28	2	ne_cseg	Count of file segments.
30	2	ne_cmod	Entries in Module Reference Table.
32	2	ne_cbnrestab	Size of non-resident name table.
34	2	ne_segtab	Offset of Segment Table.
36	2	ne_rsrctab	Offset of Resource Table.
38	2	ne_restab	Offset of resident name table.
40	2	ne_modtab	Offset of Module Reference Table.
42	2	ne_imptab	Offset of Imported Names Table.
44	4	ne_nrestab	Offset of Non-resident Names Table.
48	2	ne_cmovent	Count of movable entries.
50	2	ne_align	Segment alignment shift count.
52	2	ne_cres	Count of resource segments.
54	2	ne_psegcsum	Offset to segment checksums.
56	2	ne_pretthunks	Offset to return thunks.
58	2	ne_psegrefbytes	Offset to segment ref. bytes.
60	2	ne_swaparea	Minimum code swap area size.
62	2	ne_expver	Expected Windows version number.

Table 23: NE Header Structure

8.5 LE and LX Files

Both formats were introduced in OS/2 2.0.

The LX format supports 32-bit segments and was used under OS/2 and the Watcom Compiler/Extender (DOS).

The LE format supports mixed 16, 32, and 64-bit segments and was used under Windows 3.x, OS/2, and Windows 9x, mostly as VxD device drivers.

Headers start where e_lfanew (from the MZ structure) points to.

Offset	Size	Field	Description
0	2	e32_magic	“LX” (584Ch) or “LE” (454Ch) signature
2	1	e32_border	Byte order
3	1	e32_worder	Word order
4	4	e32_level	Linear EXE Format Level. Increments from 0
8	2	e32_cpu	Target processor i286 = 1 i386 = 2 i486 = 3
10	2	e32_os	Target operating system OS/2 = 1 Windows = 2 DOS 4.x = 3 Windows386 = 4
12	4	e32_ver	Module version
16	4	e32_mflags	Module flags
20	4	e32_mpages	Number of pages in module
24	4	e32_startobj	The Object number to which the Entry Address is relative
28	4	e32_eip	Entry Address of module
32	4	e32_stackobj	The Object number to which the ESP is relative
36	4	e32_esp	Starting stack address of module
40	4	e32_pagesize	The size of one page for this system
44	4	e32_pageshift	The shift left bits for page offsets
48	4	e32_fixupsize	Total size of the fixup information in bytes
52	4	e32_fixupsum	Checksum for fixup information
56	4	e32_ldrsize	Size of memory resident tables
60	4	e32_ldrsum	Checksum for loader section
64	4	e32_objtab	Object table offset
68	4	e32_objcnt	Object table count
72	4	e32_objmap	Object page table offset
76	4	e32_itermap	Object Iterated Pages offset
80	4	e32_rsrctab	Resource Table offset
84	4	e32_rsrccnt	Number of entries in Resource Table
88	4	e32_restab	Resident Name Table offset
92	4	e32_enttab	Entry Table offset
96	4	e32_dirtab	Module Format Directives Table offset
100	4	e32_dircnt	Number of Module Format Directives in the Table
104	4	e32_fpagetab	Fixup Page Table offset

108	4	e32_frectab	Fixup Record Table Offset
112	4	e32_impmod	Import Module Name Table offset
116	4	e32_impmodcnt	The number of entries in the Import Module Name Table
120	4	e32_impproc	Import Procedure Name Table offset
124	4	e32_pagesum	Per-Page Checksum Table offset
128	4	e32_datapage	Data Pages Offset
132	4	e32_preload	Number of Preload pages for this module
136	4	e32_nrestab	Non-Resident Name Table offset
140	4	e32_cbnrestab	Number of bytes in the Non-resident name table
144	4	e32_nressum	Non-Resident Name Table Checksum
148	4	e32_autodata	The Auto Data Segment Object number
152	4	e32_debuginfo	Debug Information offset
156	4	e32_debuglen	Debug Information length
160	4	e32_instpreload	Instance pages in preload section
164	4	e32_instdemand	Instance pages in demand section
168	4	e32_heapsize	Heap size added to the Auto DS Object (16-bit modules only)
172	4	e32_stacksize	Size of stack (undocumented)
176	196	e32_res3	Pad structure to 196 bytes (undocumented)

Table 24: LE/LX Header Structure

9 Runtime

9.1 vDOS

vDOS is an emulated DOS operating system for dd-dos, acting as the layer between the x86 interpreter and the host system. It includes a virtual shell and some tools for debugging purposes.

9.1.1 VDOS VERSION

By default, vDOS will use `DOS_MAJOR_VERSION` and `DOS_MINOR_VERSION` for its DOS version.

9.2 Interrupts

The interpreter must reproduce the interrupt procedure and have access to the host operating system services. The interrupt handling is strictly done via vDOS.

9.2.1 HARDWARE AND BIOS INTERRUPTS

Interrupts from:

- 00h to 0Fh are hardware-triggered (CPU) interrupts;
- 10h to 1Fh are software-triggered (BIOS) interrupts.

Interrupts 1Dh to 1Fh (Video, Diskette, and Video Graphics parameters) are interrupt vectors that points to data instead of instructions.

INT	**Description**
00h	Divide Overflow
01h	Single Step; Generated when the TF flag is set
02h	Non-Maskable Interrupt
03h	Breakpoint
04h	Overflow; Generated by INTO when OF is set
05h	Print screen; Sends video screen information to the printer
08h	Timer; Generates a signal every 54.92ms (≈18.2 times/sec)
09h	Generated by the keyboard whenever a key is pressed or released
0Eh	Diskette driver interface
0Fh	Printer driver interface
10h	Video driver interface
11h	Equipment Check
12h	Conventional memory size in KB
13h	Native disk I/O
14h	Serial ports
15h	Cassette interface
16h	Keyboard driver
17h	Printer I/O driver
18h	BASIC transfer control to ROM BASIC

19h	Bootstrap; Reboots the system
1Ah	Time of Day; Get or set the timer tick count
1Bh	Called by INT 9h when CTRL+BREAK is pressed
1Ch	Timer Tick; Called by INT 8h each time the timer circuit interrupts, like INT 1Bh
1Dh	Video parameters
1Eh	Diskette parameters
1Fh	Video graphics characters

Table 25: Hardware and BIOS Interrupts

9.2.2 DOS INTERRUPTS

The DOS interrupts go from 20h to 3Fh.

INT	Description
20h	Terminates program
21h	MS-DOS Services
22h	DOS Program termination address
23h	DOS Control-C/Control-Break handler
24h	DOS Critical error handler
25h	DOS Absolute disk read (except when the partition is over 32 MB
26h	DOS Absolute disk write (see note above)
27h	Terminate and stay resident
28h	DOS Idle Interrupt
29h	Fast console output
2Ah	Networking related
2Eh	WindowsNT Native API
2Fh	Printing, networking, Windows 95 functions, and some XMM (Memory Manger) function
33h	Microsoft Mouse driver

Table 26: DOS Interrupts

9.2.3 MS-DOS SERVICES

The MS-DOS Services (INT 21h) takes the AH register as a parameter, selecting the requested service.

AH	Description	MS-DOS V.
00h	Program terminate	1.0
01h	Character input	1.0
02h	Character output	1.0
03h	Auxiliary input	1.0
04h	Auxiliary output	1.0
05h	Printer output	1.0
06h	Direct console I/O	1.0
07h	Direct console input without echo	1.0
08h	Console input without echo	1.0
09h	Display string	1.0
0Ah	Buffered keyboard input	1.0
0Bh	Get input status	1.0
0Ch	Flush input buffer and input	1.0

0Dh	Disk reset	1.0
0Eh	Set default drive	1.0
0Fh	Open file	1.0
10h	Close file	1.0
11h	Find first file	1.0
12h	Find next file	1.0
13h	Delete file	1.0
14h	Sequential read	1.0
15h	Sequential write	1.0
16h	Create or truncate file	1.0
17h	Rename file	1.0
18h	Reserved	1.0
19h	Get default drive	1.0
1Ah	Set disk transfer address	1.0
1Bh	Get allocation info for default drive	1.0
1Ch	Get allocation info for specified drive	1.0
1Dh	Reserved	1.0
1Eh	Reserved	1.0
1Fh	Get disk parameter block for default drive	1.0
20h	Reserved	1.0
21h	Random read	1.0
22h	Random write	1.0
23h	Get file size in records	1.0
24h	Set random record number	1.0
25h	Set interrupt vector	1.0
26h	Create PSP	1.0
27h	Random block read	1.0
28h	Random block write	1.0
29h	Parse filename	1.0
2Ah	Get date	1.0
2Bh	Set date	1.0
2Ch	Get time	1.0
2Dh	Set time	1.0
2Eh	Set verify flag	1.0
2Fh	Get disk transfer address	2.0
30h	Get DOS version	2.0
31h	Terminate and stay resident	2.0
32h	Get disk parameter block for specified drive	2.0
33h	Get or set Ctrl-Break	2.0
34h	Get DOS flag pointer	2.0
35h	Get interrupt vector	2.0
36h	Get free disk space	2.0
37h	Get or set switch character	2.0
38h	Get or set country info	2.0
39h	Create subdirectory	2.0
3Ah	Remove subdirectory	2.0
3Bh	Change current directory	2.0
3Ch	Create or truncate file	2.0
3Dh	Open file	2.0
3Eh	Close file	2.0

3Fh	Read file or device	2.0
40h	Write file or device	2.0
41h	Delete file	2.0
42h	Move file pointer	2.0
43h	Get or set file attributes	2.0
44h	I/O control for devices	2.0
45h	Duplicate handle	2.0
46h	Redirect handle	2.0
47h	Get current directory	2.0
48h	Allocate memory	2.0
49h	Release memory	2.0
4Ah	Reallocate memory	2.0
4Bh	Execute program	2.0
4Ch	Terminate with return code	2.0
4Dh	Get program return code	2.0
4Eh	Find first file	2.0
4Fh	Find next file	2.0
50h	Set current PSP	2.0
51h	Get current PSP	2.0
52h	Get DOS internal pointers (SYSVARS)	2.0
53h	Create disk parameter block	2.0
54h	Get verify flag	2.0
55h	Create program PSP	2.0
56h	Rename file	2.0
57h	Get or set file date and time	2.0
58h	Get or set allocation strategy	2.11
59h	Get extended error info	3.0
5Ah	Create unique file	3.0
5Bh	Create new file	3.0
5Ch	Lock or unlock file	3.0
5Dh	File sharing functions	3.0
5Eh	Network functions	3.0
5Fh	Network redirection functions	3.0
60h	Qualify filename	3.0
61h	Reserved	3.0

Table 27: MS-DOS Services Functions

9.3 Internal Shell

The internal shell commands SHOULD resemble the `COMMAND.COM` commands. The commands SHOULD be case-insensitive.

9.3.1 COMMANDS (BASIC)

`DIR`

View the content of a directory.

`CD`

Display or change the current directory.

`CLS`

Clear screen.

`COPY`

Copy files.

`DEL`

Delete files.

`EXIT`

Exit shell.

`MD`

Make a directory.

`RD`

Remove a directory.

`REN`

Rename one or more files.

`TREE`

Graphically displays the directory structure.

`TYPE`

Display the content of a text file.

`VER`

View the DOS version.

9.3.2 DEBUGGING

The debugging commands have the ? character prefix to avoid conflicts with other internal shell commands.

`??`

Show debugging commands.

`?diag`

Show diagnostic runtime information. Useful for error reporting and debugging.

`?load FILE`

Manually load executable FILE.

`?p`

Toggle vCPU thread sleeping. Enabled by default.

`?panic`

Manually triggers an oops.

`?r`

Prints the states of the virtual registers.

`?run`

Manually start the vCPU as CS:IP or EIP. Useful with ?load.

`?s [MEMORY=SS:SP]`

Prints the current stack.

By default, starting from SS:SP.

Example:

```
FF10:0000 A8 02
FF10:0002 42
FF10:0003 90
```

`?u [INSTRUCTIONS=10]`

Disassemble memory in Intel-styled mnemonics.

By default, it will only disassemble 10 instructions starting from CS:IP.

Example:

```
0000FF38 A8 02  TEST 2h
0000FF40 42     INC DX
0000FF41 90     NOP
```

`?v`

Toggle verbose mode.

10 Logging

By default, verbose mode is disabled, but is enabled by default for debug builds.

If verbose is set, the logger will print all messages to the standard output.

If logging is set, the logger will write messages in a log file (dd-dos.log). Not implemented.

10.1 Log levels

- Debug: Debugging information.
  - SHOULD ONLY be available in debug builds and/or help developing dd-dos.
- Information: Purely informational.
  - e.g. VM state, file not found (in DD-DOS), default settings, etc.
- Warning: Something is probably out of place
- Error: Operation continues, but there was a mistake within dd-dos
  - e.g. Not implemented features, etc.
- Critical: Could not continue operation (oops)

A Glossary

C

CLI

Command Line Interface, describes a text-based input interface to programs to execute which either the program or the shell handles user input.

I

IMR

Interrupt Mask Register, a register within a PIC, the IMR specifies which interrupt can be ignored.

IRQ

Maskable Interrupt, a hardware interrupt that may be ignored with the IMR bit.

P

PIC

Programmable Interrupt Controller, a device that controls the flow and priorities of interrupts.

N

NMI

Non-Maskable Interrupt, a hardware interrupt that the system cannot ignore and can’t be delayed to process.