DD-DOS

A Small DOS Virtual Machine

DD-DOS Technical Manual

Version 0.6.5, 2018-03-26

Revision History

Document V.	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

Table of Content

REVISION HISTORY ..... **I**
TABLE OF CONTENT ..... **II**
TABLE OF TABLES ..... **IV**
PREFACE ..... **V**
About This Document v
Trademarks v
Disclaimer v
1 INTRODUCTION **6**
1.1 Goal 6
1.2 Notations 6
2 INITIALIZATION **7**
2.1 Command Line Interface 7
3 SETTINGS **8**
3.1 Console size 8
3.2 Date and time 8
4 X86 BYTES **10**
4.1 ModR/M Byte 10
4.2 SIB Byte 11
5 MICRO-PROCESSORS **13**
5.1 Intel 8086 13
5.1.1 Registers *13*
5.1.2 Reserved and Dedicated Memory Locations *14*
5.2 Intel i486 DX 14
6 SYSTEMS **16**
6.1 IBM PC XT 16
6.1.1 Memory Map *16*
6.1.2 I/O Port Access *16*
7 OPERATING SYSTEMS **18**
7.1 MS-DOS 18
7.1.1 Memory Map *18*
8 EXECUTABLE FORMATS **19**
8.1 Program Segment Prefix 19
8.2 COM Files 19
8.3 MZ Files 20
8.4 NE Files 21
8.5 LE and LX Files 21
9 RUNTIME **24**
9.1 DD-DOS 24
9.1.1 DOS Version *24*

9.2 Interrupts....24
9.2.1 Hardware and BIOS Interrupts....24
9.2.2 DOS Interrupts....25
9.2.3 MS-DOS Services....25
9.3 Internal Shell....27
9.3.1 Commands (Basic)....27
9.3.2 Debugging....28
10. LOGGING....30
10.1 Log levels....30
A GLOSSARY....31
C....31
CLI....31
I....31
IMR....31
IRQ....31
P....31
PIC....31
N....31
NMI....31

Table of Tables

Table 1: DD-DOS Settings....8
Table 2: Time Setting Structure....8
Table 3: Date Setting Structure....9
Table 4: ModR/M Byte Structure....10
Table 5: MOD Field Encoding....10
Table 6: REG Field Encoding....10
Table 7: ModR/M Effective Address Calculation (16-bit)....11
Table 8: ModR/M Effective Address Calculation (32-bit)....11
Table 9: SIB Byte Fields....12
Table 10: Effective Address Calculation with the SIB Byte....12
Table 11: Intel 8086 Registers....14
Table 12: Intel 8086 Dedicated/Reserved Memory and I/O Ports....14
Table 13: IBM PC XT Memory Map....16
Table 14: IBM PC XT I/O Address Map....17
Table 15: MS-DOS Memory Map....18
Table 16: DD-DOS Executable Support....19
Table 17: PSP Table Structure....19
Table 18: MZ Header Structure....20
Table 19: NE Header Structure....21
Table 20: LE/LX Header Structure....23
Table 21: Hardware and BIOS Interrupts....25
Table 22: DOS Interrupts....25
Table 23: MS-DOS Services Functions....27

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 as an aid for myself.

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

Happy reading!

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 (MS-DOS), derived from 86-DOS, was the most used DOS-based operating system from its time, making it a key product for Microsoft, growing to a rather large software company during the time.

Ever since the release of 64-bit Windows operating systems, the 16-bit emulation layer (NTVDM – WindowsNT Virtual DOS Machine) has been stripped out of the system due to x86-64 based processors not supporting the Virtual 8086 mode (VM8086, 16-bit) 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 console (`conhost`) or terminal (`tty`) window via emulation on many platforms and may eventually replace NTVDM, the WindowsNT Virtual DOS Machine.

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

1.2 Notations

This document uses the Xh notation for hexadecimal numbers (where X is the hexadecimal number), the leading zero is optional. The 0bX or bX binary notations prefixes are optinal.

All binary references are big-endian and the first bit (smallest) starts from the right side of the number.

2 Initialization

2.1 Command Line Interface

Before booting, dd-dos takes in arguments from the Command Line Interface.

Please note that only the debug builds have the D runtime CLI enabled. A list of settings can be listed with `--DRT-gcopt=help`.

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

Syntax:
```
dd-dos OPTIONS
dd-dos {-v|--version|-h|--help}
```

`--version`
Displays the version of the hypervisor, disassembler, etc., and exits.

`-h, --help`
Displays a help screen and exits.

Switches:

`-p PROGRAM`
Run a program immediately. If this option is not set, DD-DOS will start into the internal shell.

`-a ARGS`
Add arguments to `PROGRAM`. Must be after `-p`.
Not implemented.

`-v VERSION`
Set the DOS version to use. Specifying only a major version will automatically make the minor version set to 0. Not implemented.

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

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 Simple Declarative Language () format 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 fields.

Settings are currently not implemented.

Numbers must be saved as a little-endian number.

Category	Field name	Type	Default value	Description
	startapp	String	null	Default application to start.
con	width	Number	80	Initial console width.
con	height	Number	24	Initial console height.
dos	date	Number	Current date	Start date.
dos	time	Number	Current time	Start time.
dos	cwd	String	.	Starting working directory.
dos->video	useVideoModes	Boolean	true	Use or ignore video modes while running.
dos->video	startingVideoMode	Number	-1	Start DD-DOS with a specified video mode[1], a value of -1 assumes defaults.

Table 1: 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

DD-DOS saves and loads the date and time in a binary format in a little-endian encoding.

The saved time setting is a 32-bit number that complies with the following structure. The millisecond field is optional. For example, a number such as 250A06h would convert to 6:10:25.00

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

Table 2: Time Setting Structure

The saved date is a 32-bit number that complies with the following structure. For example, a number 07CB0203 would convert to 1995/Feb/03.

Offset	Size	Description and Register
0	2	Year (CX)

1 The video modes will be available in the future.

2	1	Month (DH)
3	1	Day (DL)

Table 3: Date Setting Structure

4 x86 Bytes

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

4.1 ModR/M Byte

The ModeRegister/Modifier byte is an instruction modifier under the x86 ISA, such as setting a memory mode, destination or source register, etc.

Every cell at the bottom represents a single bit (bits per fields).

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

Table 4: 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)

Table 5: MOD Field Encoding

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
110	DH	SI
111	BH	DI

Table 6: REG Field Encoding

If bit 1 is set (of the first byte of the instruction, not the ModR/M byte), the register specified in the REG field is the destination register, otherwise, source.

Effective Address Calculation					
R/M	**MOD = 00**	**MOD = 01**	**MOD = 10**	**MOD = 11**	**MOD = 11 + W[1]**
000	[SI + BX]	[SI + BX] +	[SI + BX] + D16	AL	AX

1 W (for WORD) designs the instruction's (first byte's) first bit is set.

		D8			
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 7: ModR/M Effective Address Calculation (16-bit)

The ModR/M byte's effect differ on in the 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	32-Bit D-Mode	[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 8: 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. The MOD field in the ModR/M byte that precedes the SIB byte specifies the displacement size (zero, one, or four bytes).

The SIB byte heavily depends on the ModR/M byte, since the SIB byte is only used for scaling. The displacement constant is placed after the SIB byte, then follows the immediate value.

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	-[3]	**100**	ESP
		101	EBP	**101**	See note[4]

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.
3 Illegal instruction.
4 Displacement only if MOD is cleared, otherwise EBP if MOD is set to 01 or 10.

		110	ESI	**110**	ESI
		111	EDI	**111**	EDI

Table 9: SIB Byte Fields

The table below uses n for the Scale field from the SIB byte. DISP is the displacement from the ModR/M byte (displacement follows SIB byte). In this case, B signifies BASE (Base field from SIB byte) and B8 signifies an 8-bit base register (e.g. REG8), the same applies to D with as DISPLACEMENT.

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

Table 10: Effective Address Calculation with the SIB Byte

5 Micro-processors

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.

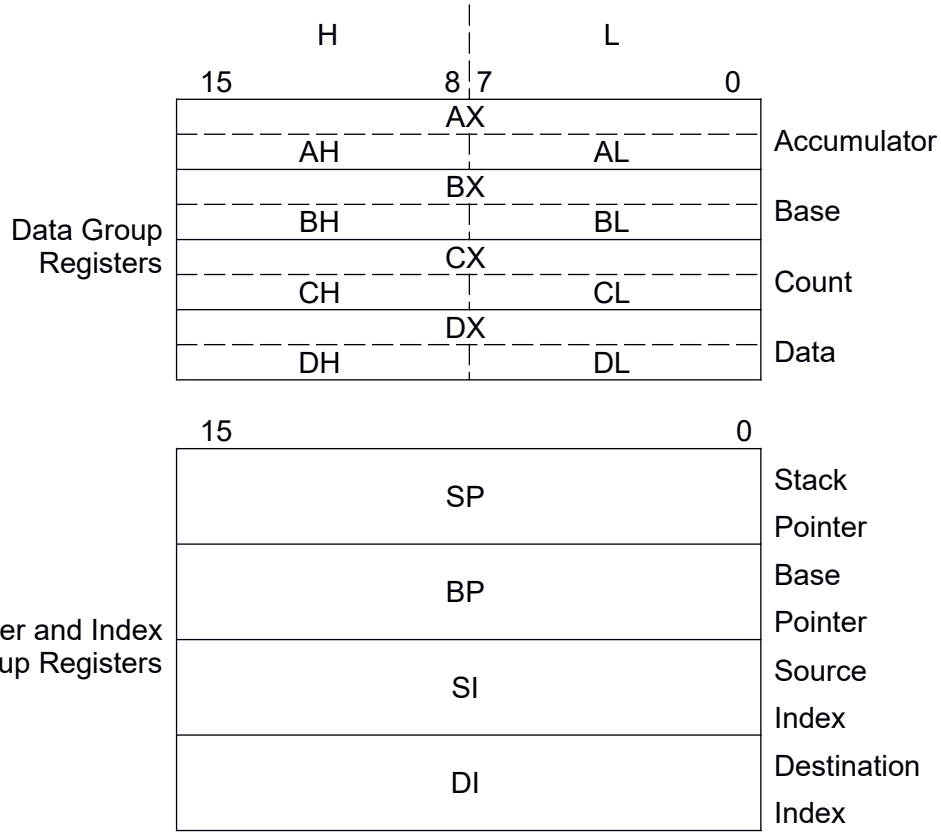
The external address bus is 20-bit wide, providing 1 MB (10_0000h) of physical address space. A linear address space is limited to 64 KB due to the 16-bit addressing and index registers. A new segmented memory model is introduced, making memory accesses into higher memory easier, which provices more than the conventional 640 KB and done 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 13 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;
- And 1 Program Counter: IP.



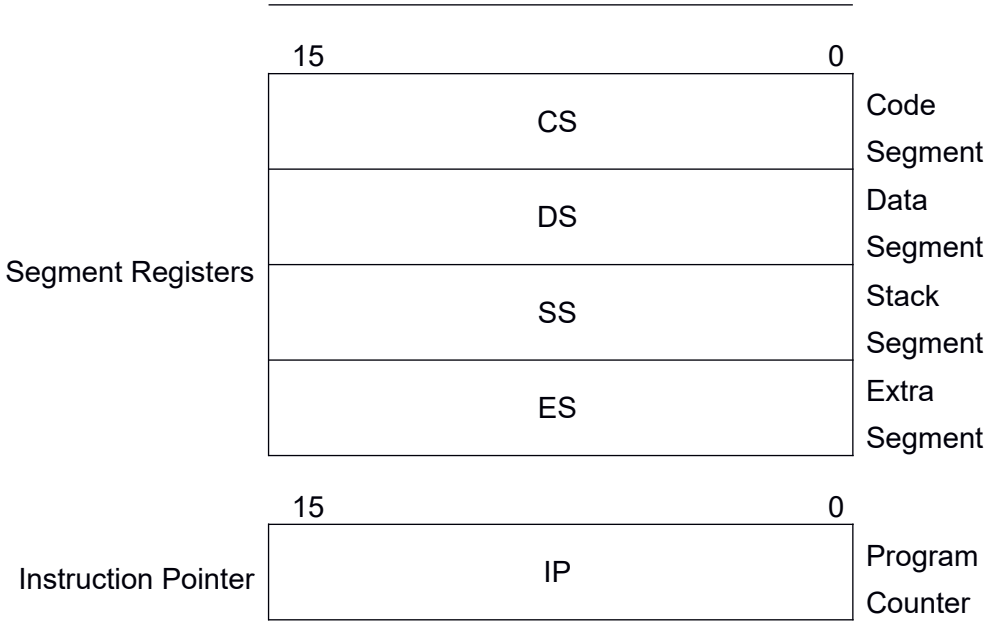


Table 11: Intel 8086 Registers

All these registers are available via properties globally.

5.1.2 RESERVED AND DEDICATED MEMORY LOCATIONS

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

Memory Locations		I/O Locations	
Reserved	FFFFFh FFFFCh	Open	FFFFh
Dedicated	FFFFBh FFFF0h		
Open	FFFEFh 80h		100h
Reserved	7Fh 14h	Reserved	FFh F8h
Dedicated	13h 0h	Open	F7h 0h

Table 12: 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.

The Intel 80486, mostly known as the i486 DX, introduced in 1989 the Floating-Point Unit, the level 1 cache (L1), and the CPUID instruction (maximum leaf 01h).

Intel i486 DX support is planned.

6 Systems

6.1 IBM PC XT

The original IBM PC was a success back in 1981, making it the barebone for future IBM PC compatible computers. It included PC-DOS with an Intel 8088 with a memory bank that can range 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 and information from the IBM 5160 Technical Reference manual.

6.1.1 MEMORY MAP

The following memory map corresponds to the IBM 5160.

Address		Function
Decimal	**Hexadecimal**	
0	0 3FFFF	256 KB of Read/Write on-board memory
256 KB	40000 9FFFF	384 KB of memory expansion
640 KB	A0000 BFFFF	128 KB of reserved memory B0000 – Monochrome B8000 – Color/Graphics
768 KB	C0000 EFFFF	192 KB ROM Expansion and Control C8000 – Fixed Disk Control
960 KB ~1023 KB	F0000 FFFFFF	64 KB Base System ROM BIOS and BASIC FE000 – BIOS Location

Table 13: 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 (Hexadecimal)	Description
000-00F	DMA Chip (8237A-5)
020-021	Interrupt 8259A
040-043	Timer 8253-5
060-063	PPI 8255A-5
080-083	DMA Page Registers
0AX[1]	Non-Maskable Interrupt Mask Register

1 At power-on, the NMI into the 8088 is masked off
Set mask: Write hex 80h to I/O Address A0h (enable NMI)
Clear mask: Write hex 00 to I/O Address A0h (disable NMI)

0CX	Reserved
0EX	Reserved
200-20F	Game Control
210-217	Expansion Unit
220-24F	Reserved
278-27F	Reserved
2F0-2F7	Reserved
2F8-2FF	Asynchronous Communications (Secondary)
300-31F	Prototype Card
320-32F	Fixed Disk
378-37F	Printer
380-38C[1]	SDLC Communications
380-389[8]	Binary Synchronous Communications (Secondary)
3A0-3A9	Binary Synchronous Communications (Primary)
3B0-3BF	IBM Monochrome Display/Printer
3C0-3CF	Reserved
3D0-3DF	Color/Graphics
3E0-3E7	Reserved
3F0-3F7	Diskette
3F8-3FF	Asynchronous Communications (Primary)

Table 14: IBM PC XT I/O Address Map

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

7 Operating systems

7.1 MS-DOS

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 15: 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 executables.

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

Table 16: 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 function 31h is specified.

The PSP table is loaded into memory before the main application (in lower memory), and the main application will use this table mostly for the command-line information.

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 17: 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) in size due to the limit of the Intel 8086.

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, ORG) 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. They start with the following header:

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 18: MZ Header Structure

If the Minimum Allocation and Maximum Allocation 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.

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

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

1 A paragraph is 16 bytes in size.

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.

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 19: 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 drivers.

Both headers start where `e_lfanew` points to with the following information:

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 20: LE/LX Header Structure

9 Runtime

9.1 DD-DOS

DD-DOS is a virtual DOS operating system for dd-dos, acting as the layer between the x86 interpreter and the host operating system.

9.1.1 DOS VERSION

By default, DD-DOS will use `DOS_MAJOR_VERSION` and `DOS_MINOR_VERSION` for its DOS version by default. The DOS version can also be specified via the command line or via the settings file.

9.2 Interrupts

The interpreter must reproduce the interrupt procedure and have access to the host operating system services.

9.2.1 HARDWARE AND BIOS INTERRUPTS

Interrupts from:

- 00h to 0Fh are hardware-triggered 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.
0Fh	Printer.
10h	Video driver.
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 21: 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) functions.
33h	Microsoft Mouse driver.

Table 22: 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 23: MS-DOS Services Functions

9.3 Internal Shell

The internal shell commands should resemble the `COMMAND.COM` commands. The commands are not case-sensitive.

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

To understand what goes on under the hood at execution time, the debugging commands provide information via the internal shell. The debugging commands have “?” as a prefix to avoid conflicts with other internal shell commands and executables.

`??`
Show debugging commands.

`?r`
Prints the states of the virtual registers.

Example:

```
EIP=00000000  (FFFF:0000)
EAX=00000000  EBX=00000000  ECX=00000000  EDX=00000000
SP=0000  BP=0000  SI=0000  DI=0000  CS=FFFF  DS=0000  ES=0000  SS=0000
FLAG= (0h)
```

`?s`
Prints the current stack.

By default, starting from SS:SP.

Example:

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

```
?u [<NBInts>]
```
Disassemble memory in Intel-styled mnemonics.

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

Example:

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

10. Logging

By default, the Verbose mode is disabled (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 avaiable 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, used to denote the interface a user can use via a command-line prompt.

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 cannot be delayed).