SUMMARY OF 86-DOS 1.0 ENHANCEMENTS

* Each file in the disk directory has a date which is displayed with the DIR command. This date represents the last time the file was written to.

* The system may execute batch jobs: Any number of commands can be strung together in a file, which can then be executed in sequence when desired. At the time execution of the file of commands is started, parameters may be included to alter the exact effect.

* The console, printer, and auxiliary I/O device may be referenced with file names. This provides a natural extension for commands and programs that do file I/O, allowing them to communicate through the I/O devices.

* Files may now be manipulated on a byte-by-byte basis. For example, after editing a file with EDLIN, its size as displayed by the DIR command is correct to the exact byte.

* The I/O system now allows disk sector sizes other than 128 bytes without any additional programming effort. As a result, the larger sector sizes normally associated with hard disks or double density floppy disks (256 to 1024 bytes) is supported directly.

* Full double density operation is now supported with the Tarbell DD disk controller. Using 1K sectors and double-sided drives gives 1.2 Mbyte per drive.

* DEBUG has been substantially upgraded to include the following features: All addresses now use a segment:offset format; disassembly is now provided; HEX files are loaded with automatic binary conversion; areas of memory can be compared for equality; the Go and Trace commands can set CS and IP to any desired value before executing the program under test.

* EDLIN can now edit files too large to fit into memory, and it can be aborted without saving changes if desired.

* The assembler is much faster and provides a new, complete handling of the ESC code for generating 8087 opcodes.

* HEX2BIN allows a load offset to be specified.

* RDCPM will now accept the "wild card" characters ("?" and "*") to copy multiple files in one command. It can also be used to list the directory of the CP/M disk.

SCP 86-DOS™

Version 1.0 Addendum

OPERATING SYSTEM

New Directory Format

The size of each entry in the disk file directory has been expanded from 16 bytes to 32 bytes. This allows keeping more information about each file with plenty of room for future expansion (e.g., passwords). However, earlier versions of 86-DOS are NOT capable of reading disks which use the new directory entry. Version 1.0 IS capable of reading disks which use the old directory entry. This capability may not be maintained in future releases of 86-DOS, so it is strongly recomended that all disks using the old format be copied into the new format. The size of directory entry is set by the CLEAR command.

One of the effects of the new directory entry size is that files are no longer limited to 16 Mbytes in size. Files may now be as large as the disk they are on. Disk drives are limited to 1 Gbyte each (2^30 bytes, about 1 trillion).

A date field is also maintained in the new directory entry. Every time a file is written to, the current date is recorded in its directory entry. This date is displayed by the DIR command and is accessable to user programs through the File Control Block. When the system is first started, it prompts the user for the present date which will be used until the system is restarted.

I/O Device Names

86-DOS supports three I/O devices other than disks: the console, the auxiliary, and the printer (output only). These devices may now be referred to as files with file names. The names are CON, AUX, and LST or PRN. The names may be followed by a colon if desired, or they may be given any extension, which will be ignored.

All devices are treated as ASCII files, with Ctrl-Z (1A hex) used as the end-of-file mark. When reading CON there is no prompt, but input is fully buffered (nothing happens until you hit <return>), and the special editing functions may be used. End of file is recognized as a Ctrl-Z in the first position in the line. The program reading CON as a file may take time to digest each line, so an indicator of "readiness" is built in. A new line may be typed in if a linefeed has been echoed since the last carriage return. Once entering a line, there is no speed limit because input is buffered by lines.

One of the uses of these device names is to use COPY to transfer ASCII files between machines using the AUX port. When parallel I/O is used, extremely high data rates may be obtained (many times faster than floppy disk transfer speeds!). Note that only ASCII data may be used, because the first Ctrl-Z (1A hex) encountered will be considered an end-of-file mark.

Another example of an interesting use is the command

```
ASM  CON.AZA
```

which will effectively put the assembler in "direct mode". Once the command is given and the assembler has signed on, a program may be typed in line by line. Full use of all assembler facilities (labels, ORGs, etc.) is available. When done, entry is terminated with Ctrl-Z and <return>. The assembler now enters the listing pass where it expects to re-read the source code. However, it is not necessary to type the program in again. Instead, each time <return> is typed the address and object code of each line of the program will be displayed. This method may be handy for quick assembly of very short programs.

One last example of using device names is for printing a number of files. COPY may be used to transfer files directly to PRN (or LST). Ambiguous souce file names may be used so several files may be printed with one command. Note that TABs (09 hex) will NOT be expanded but instead sent directly to the printer.

Correction to Manual

Page six of the Programmer's Manual, Interrupt Table Usage, refers to interrupt types "32 to 63 hex". These are actually decimal numbers, and in hex would be interrupts 20 to 3F.

COMMAND INTERPRETER

Batch Processing

The Command Interpreter will now execute a sequence of commands from a file. The file may have any name with the extension ".BAT". To execute the commands in the file, simply type the file name (without the extension), along with any parameters as may be appropriate. For example, you may type

```
NEW  B:
```

which is a batch file provided on the disk to make copies of the disk. COMMAND will first search for NEW.COM, which it would assume is an excutable binary program. Failing to find it, it will then search for NEW.BAT, which it will find and to which it will assign the parameter "B:".

Up to nine parameters may passed from the input line to the batch file. Each parameter is substituted for a special symbol in the batch file before the each command is excuted. The special symbol is the percent symbol, "%", followed by a digit. "%0" refers to the first thing typed on the input line, which is the name of the batch file itself. "%1" refers to the first parameter typed on the input line, up to "%9" which refers to the ninth. If it desired to have the "%" symbol itself appear in the batch file, it is represented by two consecutive "%".

For example, suppose the file NEW.BAT was prepared by EDLIN and contained the following:

```
CLEAR  %1
SYS  %1
COPY  A:*.COM  %1
PAUSE  To make more copies, insert new disk in drive %1
%0  %1
```

In response to the sample command "NEW B:" shown above, the following sequence of commands would be executed:

```
CLEAR  B:
SYS  B:
COPY  A:*.COM  B:
PAUSE  To make more copies, insert new disk in drive B:
NEW  B:
```

PAUSE is a new command which simply prompts for a carriage return to continue. This allows time to change disks (or any other operation) before continuing with the batch job. The appearance of NEW as the last line causes NEW.BAT to be excuted again. If the disk in drive B was changed during PAUSE, then a second copy can be made.

Batch jobs may not be nested. One job may start another (as shown above), but all record of the first job will then be lost.

If a command or program that is part of a batch job is aborted with Ctrl-C, the user will be asked if the batch job should also be aborted. A response of "N" will allow the batch job to continue, "Y" will cause it to abort.

Copy Command

The Copy command has been streamlined somewhat when copying between drives. Also, the date of the copy is assigned to be the same as the date of the original.

Addresses

All addresses now use a distinct segment and offset. When displayed, an address is of the form

```
xxxx:yyyy
```

where xxxx is the segment, and yyyy is the offset within the segment. The corresponding physical address in the 8086's 20-bit address space is figured as (segment * 10H) + offset:

```
xxxx0  (segment * 10H)
 yyyy  (offset)
-----
zzzzz  (20-bit physical address)
```

When entering an address, 1) the segment may be specified explicitly, using the same format as displayed; 2) a segment register designation (CS, DS, ES, SS) may be used, to specify the current value in the register save area; or 3) the segment may be omitted, in which case a default (dependent on the command) will be used. Some examples of addresses in each category:

1) 35B:100 0:C000

2) DS:6 CS:100

3) F00 1CB7

Note that in forms one and two, no spaces may occur around the colon.

To reference an arbitrary 5-digit physical address, simply put a colon between the fourth and fifth digits. For example, location 12345 hex may be referenced with 1234:5. This works out like this:

```
12340  (segment * 10H)
 0005  (offset)
-----
12345  (physical address)
```

When the segment is not specified, the default for most commands is the data segment, i.e., the current value in the register save area for the DS register. The exceptions to this are the Go and Unassemble commands, which use the code segment.

Register Display

The first two lines of a register display are unchanged. A third line (or possibly more) has been added showing the next instruction, in both

Load and Write Commands

The command to Load (read) and Write absolute sectors have not been changed, but the I/O system calls used to implement them have. The absolute sector number now refers to physical disk sectors (of whatever size has been defined by the I/O system), not 128-byte records. The length is also a count of physical sectors, not 128-byte records. Care should be taken whenever these commands are used (particularly the Write command), to ensure the desired effects are achieved (rather than simply destroying the disk).

Load and Write as applied to named files have been enhanced. These commands now accept one parameter, an <address>, which is the address at which to load or from which to write the file. If the address is omitted, the default is CS:100H. CX now represents the size of the file in bytes, rather than 128-byte records.

The file load command now also allows loading HEX files. This mode is automatically invoked if the file name has the extension ".HEX". If no address is specified, the file is loaded in the Code Segment at the addresses encoded in the HEX file. If an <address> is present, the file is loaded in the specified segment, and the addresses encoded in the HEX file are added to the offset to determine the load address. Upon completion, CX has the length of the file relative 0 in the load segment, which is the same as the highest address loaded plus one.

Go and Trace Commands

A feature has been added to these commands to allow setting CS and IP before the trace or execution. The first parameter of the command may be an equals sign, "=", followed by an <address>, which is the value to be assigned to CS and IP. This parameter may then be followed by the usual breakpoint list or trace count, as appropriate. For example,

```
G =117, 3CB:15
```

sets IP to 117 hex, leaves CS unchanged, and sets a breakpoint at 3CB:15 before starting execution.

If a program is executing under the Go command, then Ctrl-C may be used to halt the executing program during input or output. This will occur while the operating system is executing an I/O system call for the program. All registers will be restored to the same value they had just before the program made the system call, except CS:IP will point to the first instruction after the system call.

hexadecimal and symbolic form. Also, if the instruction makes a memory reference, the segment and effective address are displayed along with the current value of that location.

Unassemble Command

Formats:

```
U
U <address>
U <range>
```

This command provides disassembly of portions of a program. If no parameters are specified, the command continues from the last disassembled line. If no length or ending address is specified, a length of 20H is assumed.

Whenever the Register command is used, the current instruction pointed to by CS:IP is disassembled. This also sets the default address of the Unassemble command to the following instruction. Thus the instructions at and after the current CS:IP may be reviewed at any time by following a Register command with an Unassemble command with no parameters.

Default Dump Address

The Dump command may now be used with no parameters, when it will display the 80 hex bytes immediately following the last byte displayed in a previous Dump command. Initially, the default dump address is CS:100.

Compare Command

Format:

```
C <range> <address>
```

The Compare command performs a byte-by-byte comparison of the specified range with the data starting at the specified address. For each pair of bytes that don't match, the following is displayed on one line: The segment and offset of the first byte; the value of the first byte; the value of the second byte; the segment and offset of the second byte. Note that the display operation causes the locations to be read a second time; it is thus possible for the bytes to be displayed with the same value if a memory failure has occured such that their values could change between the first and second readings. If every byte matches, there is no display.

HEX2BIN

HEX2BIN now allows a second parameter to be entered after the name of the file to be converted from hex to binary. This parameter is a signed hex number which represents an offset to be added to the load addresses encoded in the hex format. To use this offset, it is necessary to have an understanding of just what HEX2BIN does:

First, as large a segment as possible (limited to 64K and available memory) is set aside as the load segment. This segment is filled with zeros. Then the HEX file is read piece by piece and decoded. Each time a load address is encountered in the HEX file, the offset is added to it modulo 64k and loading continues at that point. The highest location loaded is also kept track of. The loading processes is aborted if the load address (with offset added to it) exceeds available memory. After loading, the file is saved starting at location 0 in the segment, up to the highest location loaded (to the exact byte).

Normally, a program intended to become a ".COM" file will be assembled with a "PUT" address of 100H, since this is where it will execute. However, the program must actually be written starting right at the beginning of the COM file because the file itself is loaded at 100H. This requires a load offset of -100 hex, which is the default offset if none is specified. That is, if no offset is specified (no second parameter to HEX2BIN), the HEX file will have 100H subtracted from the encoded load addresses; since these addresses start at 100H (from the PUT 100H in the source code), the file will be loaded (and saved) at location 0. When the file is later loaded as a command, it will load at 100H which is its proper execution address.

The offset parameter is particularly useful when converting a file not intended as an 86-DOS COM file. The parameter has an optional "+" or "-", then a hex number with one to four digits. Note that if an offset of 0 is desired, it must be specified explicitly since the default is -100H.

The date of the COM file produced by HEX2BIN is the same as the date of the original HEX file.

ASSEMBLER

The assembler now uses considerably more memory buffering resulting in a substantial improvement in speed. The HEX and PRN files it generates will have the same date as the original ASM file. The ESC opcode has been fully upgraded to allow generating all possible 8087 opcodes. There are now two forms:

```
ESC     VALUE1,[ADDR]

ESC     VALUE1,VALUE2
```

VALUE1 is a number in the range 0 to 63 which is internally represented by 6 bits. The leftmost 3 bits form the last part of the first byte of the ESC opcode (the first 5 bits are always 11011). The rightmost 3 bits of VALUE1 form the middle section (bits 3,4,5) of the second byte of the ESC opcode. The rest of the second byte of the ESC opcode is determined by the second operand.

If the second parameter is [ADDR], then the second byte of the ESC opcode is set up for the correct addressing mode and possibly a one or two byte displacement is appended to the two opcode bytes. Thus this form may vary from 2 to 4 bytes. Graphically, the first two bytes are:

```
1 1 0 1  1 x x x        m m y y  y r r r
```

Where xxxyyy = VALUE1, mm = MOD field, rrr = R/M field

If the second parameter is VALUE2, VALUE2 must be in the range 0 to 7, which is internally represented by 3 bits. These 3 bits are placed in bits 0, 1, and 2 of the second byte of the ESC opcode and bits 6 and 7 of are both 1. Thus the two bytes look like this:

```
1 1 0 1  1 x x x        1 1 y y  y z z z
```

Where xxxyyy = VALUE1, zzz = VALUE2

EDLIN

EDLIN can now edit files too large to fit into memory. When starting an edit of a large file, EDLIN will load as much of the file as fills 3/4 of available memory (but always ending with a full line). Two new commands have been added to manipulate which portion of the file is present in memory.

Append Command

This command causes additional lines of text to be read from disk into memory. The format is

<line> A

where <line> is the number of lines to be appended to the text in memory. If <line> is omitted, the default is the maximum number of lines that will fit in memory. If there is not enough memory to hold all the lines requested, then as many whole lines as will fit are appended and a Memory Full message appears. When this happens, no more lines may be appended until some are written out with the Write command. If there aren't as many lines left in the file as were requested, the message "End of input file" appears.

Write Command

This command writes line of text to the disk from memory. The format is

<line> W

where <line> is the number of lines to write; if not specified, all lines before the cursor are written. Once a line has been written, it is not available to be edited again except by ending the editing session then restarting it. Writing lines causes the line numbers to shift in the same manner as line deletion, and the cursor will be moved to line 1.

Quit Command

This command allows ending the editing session WITHOUT saving any changes that may have been made to the file. The original input file will be unchanged but its backup will be deleted. To prevent losing a lot of hard work, this command will prompt to be sure there is no mistake; a "Y" response is required to finish the command.

File Lengths

Files produced by EDLIN are the exact byte length of the text that was being edited plus one end-of-file mark. This length is properly displayed by the DIR command of the Command Interpreter.

RDCPM

RDCPM now accepts "wild card" characters ("*" and "?") in file names. This makes RDCPM almost identical to COPY in use. A typical command might be

```
RDCPM  B:*.BAS
```

which will copy all BASIC programs from the CP/M disk in drive B to the default drive. Renaming files while copying is also simplified. For example:

```
RDCPM  B:PROGRAM.A86  *.ASM
```

copies only one file and changes the extension from ".A86" to ".ASM". The "*" in the second file name simply eliminates the need to type the name over again.

A special form of RDCPM parameters allows the directory of the CP/M disk to be displayed. The formats are:

```
RDCPM  DIR  <drive>   or   RDCPM  DIR  <filename>
```

The first format displays the entire directory, with four file names per line. The second displays only those files which match the given file name (wild card characters allowed). Note that in the second form, the file name will need the disk drive specifier of the CP/M drive.

I/O SYSTEM

The following changes have been made in the requirements of the I/O system:

FLUSH Routine

This routine, entered at 041B hex, is no longer used. In its place, at the same location, is a new routine called DSKCHG. This new routine takes as input a disk drive number in AL (starting with zero). It returns

AH = -1 if disk has been changed.
AH = 0 if it is not known whether the disk has been changed.
AH = 1 if disk could not have been changed.

AL may be destroyed by the routine but no other registers may be affected. The purpose of this routine is to minimize unnecessary re-reading of disk directory information.

Floppy disk systems with no way to know if the disk has been changed will simply return AH = 0 whenever this routine is called. Some floppy disk drives provide a disk change signal, which simply latches the fact that the drive door has been opened since the last disk access. Another way to tell is if the head on the drive is still loaded from the last disk command, then one may assume the disk has not been changed. (In this case, the head not loaded does not mean the disk has been changed, it means unknown.) A non-removable hard disk should always return that disk is not changed.

Initialization Tables

The formats of the initialization tables are quite different and should be much easier to work with. The entries of the main initialization table are as follows:

1. 1 Byte. The number of disk drives, N.

2. 2*N Bytes. For each drive, the address (two bytes) of that drive's Drive Parameter Table (DPT)--see below. Similar drives may share a DPT.

3. 2 Bytes. The maximum amount of memory 86-DOS may use for buffering. This number will be ignored if it is less than the minimum required by 86-DOS. 86-DOS version 1.0 presently ignores this field and uses the minimum amount.

4. 2 Bytes. The amount of stack space required by the I/O system. 86-DOS version 1.0 presently ignores this field and provides a fixed 30 bytes (15 levels) of stack space. If this is insufficient, the I/O routine must swap to an internal stack.

The Drive Parameter Table, or DPT, has the following entries:

1. SECSIZ. 2 Bytes. The size, in bytes, of the physical disk sector. The minimum value is 32 bytes, the maximum practical value is 16K. This number need not be a power of two.

2. CLUSSIZ. 1 Byte. The number of sectors in an allocation unit. This number must be a power of two. This limits it to the values 1, 2, 4, 8 ,16, 32, 64, and 128. By making the allocation unit small, less disk space is wasted because the last allocation unit of each file is only half full on the average. By making the allocation unit large, less space is taken up both on the disk and in memory for the File Allocation Table. A good choice is to make the allocation unit approximately equal to the square root of the disk size (to the nearest power of two). For example, a standard floppy disk with 256K would use an allocation unit of 512 bytes, or 4 physical sectors. A 2D floppy disk with 1K sectors has 1.2 Mbytes, and would use an allocation unit of 1K, or 1 physical sector.

3. RESSEC. 2 Bytes. The number of reserved sectors at the start of the disk. At least one sector is usually reserved for a disk bootstrap loader and more may reserved to place the I/O system or all of 86-DOS in this reserved area.

4. FATCNT. 1 Byte. The number of File Allocation Tables. This is normally two, to provide one backup.

5. MAXENT. 2 Bytes. The number of directory entries. This may be any number less than 4096. For maximum efficiency, however, it should be a multiple of the number of directory entries that can fit in one physical sector, at 32 bytes per directory entry.

6. DSKSIZ. 2 Bytes. The number of physical disk sectors. Being represented with only 16 bits, this number clearly must be less than 64K. If a large disk has more physical sectors than this, the size of the physical sector seen by 86-DOS must be increased by using multiples of the physical sector. Every time the I/O system documentation says "physical sector", consider this to mean, for example, two physical sectors. Then the size of this new "physical sector", SECSIZ, is twice as big as before, DSKSIZ is half as big, and the READ and WRITE routines must work in terms of these new sectors.

READ and WRITE Routines

The requirements of these routines are unchanged, except that instead of transferring a 128-byte record, they transfer physical sectors as defined by the SECSIZ field of the DPT. As internal buffering is no longer necessary in the I/O system, the directory write flag in AH is no longer provided on disk writes.

86-DOS version 1.0 File Control Block

The only differences between the File Control Block for version 1.0 and earlier versions is that bytes 14-21 which were reserved for system use are now defined for use by user programs, and an extra byte may be added to the random record field if small (less than 64 bytes) records are used.

BYTE 0-Drive Code. Zero Specifies the default drive, 1=drive A, 2=drive B, etc. Note that other functions which use a drive number use 0=drive A, 1=drive B, etc.

BYTES 1-8 - File Name. If the file name is less than 8 characters, the name must be left justified with trailing blanks.

BYTES 9-11 - Extension. If less than 3 characters, must be left justified with trailing blanks. May also be all blanks.

BYTES 12-13 - Current Block. This word (low byte first) specifies the current 16K block, relative the start of the file, in which sequential disk reads and writes occur. If zero, then the first 16K of the file is being accessed; if one, then the second 16K; etc. Combined with the current record field, byte 32, a particular 128-byte record is identified.

BYTES 14-15 - Record size in bytes. Automatically set to 128 by the operating system when the file is opened but can be set to any size desired for file I/O. Notice that if a size other than 128 bytes is to be used this field must be set after the file is opened because 86-DOS sets the record size to 128 when the file is opened. If the record size used is less than 64 bytes, the random record field (bytes 33-36) is increased to 4 bytes.

BYTES 16-19 - Size of file in bytes. This 32 bit number (low byte first) indicates the file size in bytes and may be examined by the user program but must NOT be changed.

BYTES 20-21 - This 16-bit field (low byte first) is the date. The 5 least significant bits are the day, the next 4 are the month, the last 7 are the year minus 1980 (i.e., 1980=0). The date is always set to the current date when data is written to the file. To change the date, set it to the desired value after writing to the file but before closing the file. Some data must be written to the file or the new date set by the user program will not be used.

BYTES 22-31 - Reserved for system use once the file is opened and until it is closed.

BYTE 32 - Current Record. Identifies the record within the current 16K block that will be accessed with a sequential read or write function.

BYTES 33-36 - Random Record. This 24-bit or 32-bit number (low byte first) need be present only when the file is accessed with a random read or write function. It is the position in the file of a record. If the record size (as defined in bytes 14-15) is less than 64 bytes, then this field will be 4 bytes long (bytes 33-36). If the record size is 64 bytes or larger, then this field will be three bytes long (bytes 33-35). The size of the FCB will be either 36 or 37 bytes depending on the size of this field.

Notice that there are two ways to address a record within a file. The Current Block and Current Record fields together address a record when the file is accessed with the sequential read and write functions. The Random Record field addresses a record when the file is accessed with the random read and write functions. The appropriate fields may be set before either a sequential or random transfer to select the next record to be accessed.