Programmer's Manual

Version 0.3

86-DOS™

Disk Operating System for the 8086

Preliminary
# 86-DOS™

**Programmer's Manual**

## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Guide</td>
<td>3</td>
</tr>
<tr>
<td>Operating System Calls</td>
<td>3</td>
</tr>
<tr>
<td>Summary of 86-DOS Functions</td>
<td>4</td>
</tr>
<tr>
<td>Interrupt Table Usage</td>
<td>6</td>
</tr>
<tr>
<td>Requesting a Function</td>
<td>7</td>
</tr>
<tr>
<td>Simple Device I/O Functions</td>
<td>7</td>
</tr>
<tr>
<td>Miscellaneous Functions</td>
<td>15</td>
</tr>
<tr>
<td>Using Operating System Functions</td>
<td>16</td>
</tr>
<tr>
<td>Running a User Program</td>
<td>17</td>
</tr>
<tr>
<td>Customizing 86-DOS</td>
<td>19</td>
</tr>
<tr>
<td>Setting the Special Editing Commands</td>
<td>19</td>
</tr>
<tr>
<td>Customizing the I/O Section</td>
<td>20</td>
</tr>
<tr>
<td>Bootstrap Loader Listing</td>
<td>26</td>
</tr>
<tr>
<td>I/O Section Listing</td>
<td>29</td>
</tr>
</tbody>
</table>

COPYRIGHT 1980 by Seattle Computer Products Inc.
All rights reserved.
Programming Guide

Operating System Calls

The purpose of the operating system core is to provide a high-level, hardware independent interface between a user program and its hardware environment. Most functions that the user may request can be grouped into two categories: simple device I/O and disk file I/O.

The simple I/O functions are:

Input console character
Output console character
Input from auxiliary
Output to auxiliary
Output to printer
Output character string to console
Input Buffered line from console
Check console for input character ready

The disk I/O functions include:

Reset disk system
Select default disk
Scan disk directory
Create file
Open file
Close file
Delete file
Rename file
Read/Write file record(s)
Set disk transfer address
## Summary of 86-DOS Functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Function</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Program terminate</td>
<td>None</td>
<td>None</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>Console Input</td>
<td>None</td>
<td>AL = Character</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>Console Output</td>
<td>DL = Character</td>
<td>None</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Auxiliary Input</td>
<td>None</td>
<td>AL = Character</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>Auxiliary Output</td>
<td>DL = Character</td>
<td>None</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>Printer Output</td>
<td>DL = Character</td>
<td>None</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Direct Console I/O</td>
<td>DL = OFFH or DL = Character</td>
<td>AL = Char. if ready or AL = 0 if not ready</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>Output String</td>
<td>DS:DX = String</td>
<td>None</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>Input String</td>
<td>DS:DX = Buffer</td>
<td>None</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>Check Console Status</td>
<td>None</td>
<td>AL = OFFH if ready or AL = 0 if not ready</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>Disk System Reset</td>
<td>None</td>
<td>None</td>
<td>11</td>
</tr>
<tr>
<td>14</td>
<td>Select Default Drive</td>
<td>DL = Drive</td>
<td>AL = No. of drives</td>
<td>11</td>
</tr>
<tr>
<td>15</td>
<td>Open File</td>
<td>DS:DX = FCB</td>
<td>AL = Error flag</td>
<td>11</td>
</tr>
<tr>
<td>16</td>
<td>Close File</td>
<td>DS:DX = FCB</td>
<td>AL = Error flag</td>
<td>11</td>
</tr>
<tr>
<td>17</td>
<td>Search for First</td>
<td>DS:DX = FCB</td>
<td>AL = Found flag</td>
<td>12</td>
</tr>
<tr>
<td>18</td>
<td>Search for Next</td>
<td>DS:DX = FCB</td>
<td>AL = Found flag</td>
<td>12</td>
</tr>
<tr>
<td>19</td>
<td>Delete File</td>
<td>DS:DX = FCB</td>
<td>AL = Found flag</td>
<td>12</td>
</tr>
<tr>
<td>20</td>
<td>Sequential Read</td>
<td>DS:DX = FCB</td>
<td>AL = Error flag</td>
<td>12</td>
</tr>
<tr>
<td>21</td>
<td>Sequential Write</td>
<td>DS:DX = FCB</td>
<td>AL = Error flag</td>
<td>12</td>
</tr>
<tr>
<td>22</td>
<td>Create File</td>
<td>DS:DX = FCB</td>
<td>AL = Error flag</td>
<td>12</td>
</tr>
<tr>
<td>23</td>
<td>Rename File</td>
<td>DS:DX = Modified FCB</td>
<td>AL = Found flag</td>
<td>13</td>
</tr>
<tr>
<td>25</td>
<td>Get Default Drive</td>
<td>None</td>
<td>AL = Default drive</td>
<td>13</td>
</tr>
<tr>
<td>26</td>
<td>Set Disk I/O Address</td>
<td>DS:DX = I/O address</td>
<td>None</td>
<td>13</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Registers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocation Address</td>
<td>None</td>
<td>DS:BX = Address, DX = Disk size, AL = Block size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameter Address</td>
<td>None</td>
<td>DS:BX = Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Read</td>
<td>DS:DX = FCB</td>
<td>AL = Error flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Write</td>
<td>DS:DX = FCB</td>
<td>AL = Error flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get File Size</td>
<td>DS:DX = FCB</td>
<td>AL = Error flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get File Address</td>
<td>DS:DX = FCB</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set Vector</td>
<td>DS:DX = Vector address, AL = Interrupt type</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create Segment</td>
<td>DX = Segment number, None</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Block Read</td>
<td>DS:DX = FCB, CX = Record count</td>
<td>AL = Error flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Block Write</td>
<td>DS:DX = FCB, CX = Record count</td>
<td>AL = Error flag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parse File Name</td>
<td>DS:SI = Input line, SI updated, ES:DI = FCB, AL = 0 (no pre-scan), AL = 1 (scan separators)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Interrupt Table Usage**

The first 1K of memory, absolute address 00000 to 003FF hex, is reserved by the 8086 for the interrupt table. Within this table, locations 00080 to 000FF, which correspond to interrupt types 32 to 63 hex, are reserved for 86-DOS. Specific interrupt types have been defined as follows:

32 - Program terminate. The Terminate and Ctrl-C Exit addresses are restored to the values they had on entry to the terminating program. All file buffers are flushed, but files which have been changed in length but not closed will not be recorded properly in the disk directory. Control transfers to the Terminate address.

33 - Function request. See "Requesting a Function", below.

34 - Terminate address. On entry to a program, this is the address to which control will transfer when the program terminates. This address is copied into low memory in the program segment when it is created by Function 38. A program may change this address, but this does not affect what happens when it terminates, since the Terminate address is restored from the copy in the program segment. If the program executes a second program, it must set the Terminate address to the location that the second program will transfer to on termination.

35 - Ctrl-C Exit address. If the user at the console types Ctrl-C during console input or output, an interrupt type 35 hex is executed. If the Ctrl-C routine preserves all registers, it may end with a return-from-interrupt instruction (IRET) to continue program execution. If the Ctrl-C handler does nothing but an "IRET", Ctrl-C will appear to have no effect.

If the program executes a second program which itself changes the Ctrl-C Exit address, then on termination of the second program and return to the first, the Ctrl-C address is restored to value it had before the second program changed it.

36 - Hard Disk Error Exit address.

37 - Absolute disk read. Control transfers directly to the I/O system disk read routine. On return, the original flags are still on the stack (put there by the INT instruction), which is necessary because return information is passed back in the flags. Be sure to pop the stack to prevent uncontrolled growth.

38 - Absolute disk write. See above.
Requesting a Function

The user program requests a function by putting a function number in the AH register, possibly setting another register according to the function specifications, and performing an interrupt type 33. The user's stack must have a total of 16 levels (32 bytes) of space available before performing the interrupt, which will insure compatibility with future multi-user versions of 86-DOS. When 86-DOS takes control it saves all the user's registers except AL and switches to an internal stack. Thus all registers, including the flags but excepting AL, will be unchanged on return unless noted otherwise in the function specification.

Those functions whose numbers are 36 or less are also available through a different call mechanism. The function number is placed in the CL register, any other registers are set in their usual way according to the function specifications, and a normal 3-byte (intra-segment) "call" is made to location 5 in the current code segment. Register AX is always destroyed by this calling method, but otherwise it is the same as the normal (interrupt) method. This form is provided to simplify translation of 8080/280 programs into 8086 code, and is not recommended for new programs.

Simple Device I/O Functions

1 - Console input. Waits for a character to be typed on the console, then echos the character (as in Function 2) and returns it in AL. The character is checked for a control function as described in Function 2 below.

2 - Console output. The character in register DL is output to the console. The parity bit (bit 7) must be zero unless a special terminal function is desired. Tabs are expanded in columns of 8. Rubout (7F hex) is output but is not counted in tab counting. After output, the console is checked for a control function:

- Ctrl-S suspends everything until any key is typed.
- Ctrl-P sends all console output to the printer also.
- Ctrl-N stops sending output to the printer.
- Ctrl-C causes an interrupt to the Ctrl-C address.

3 - Auxiliary input. Waits for a character from the auxiliary input device, then returns that character in AL.
SIMPLE DEVICE I/O FUNCTIONS (Continued)

4 - Auxiliary output. The character in DL is output to the auxiliary output device.

5 - Printer output. The character in DL is output to the printer.

6 - Direct Console I/O. If DL is FF hex, then AL returns with a console input character if one is ready, otherwise 00. If DL is not FF hex, then DL is assumed to have a valid character which is output to the console.

9 - Print string. On entry, DS:DX must point to a character string in memory terminated by a "$" (24 hex). Each character in the string will be output to the console in same form as Function 2, including subsequent status check.

10 - Buffered console input. On entry, DS:DX point to an input buffer. The first byte must not be zero and specifies the number of characters the buffer can hold. Characters are read from the console and placed in the buffer beginning with the third byte. Reading the console and filling the buffer continues until carriage return is typed. If the buffer fills to one less than maximum, then additional console input is ignored until a carriage return is received. The second byte of the buffer is set to the number of characters received excluding the carriage return (0D hex), which is always the last one.

A number of control functions are recognized while reading the console:

Tab, Ctrl-S, Ctrl-P, Ctrl-N, Ctrl-C have the same effects as listed under Function 2.

Rubout, delete, backspace, Ctrl-H (7F hex or 08 hex): Backspace. Removes the last character from the input buffer and erases it from the console.

Linefeed, Ctrl-J (10 hex): Physical end-of-line. Outputs a carriage return and linefeed but does not effect the input buffer.

Ctrl-X (18 hex): Cancel line. Outputs a back slash, carriage return, and linefeed and resets the input buffer to empty. The template used by the special editing command is unchanged.

SPECIAL EDITING COMMANDS. A number of special editing commands are available to the user entering a line at the console. All of these involve a "template", which is a valid input line available to the user for modification. There are two ways to obtain a template.

If the input buffer already contained a valid input line on entry to Function 10, then this is a template. A valid input line is one in which the character
count at the second byte of the buffer is less than the buffer length, and a carriage return (0D hex) immediately follows the text in the buffer. Note that a buffer that has previously been used for input and has not been modified will meet these requirements.

The user at the console may also create a template. One of the editing commands is to convert that part of the line entered so far into the template, and restart the line entry. This allows an error near the start of a line to be corrected without retyping the rest of the line.

Each editing command is selected by typing ESCAPE and a letter. Since many terminals provide keys which produce such an "escape code" with a single keystroke, the letter used after the ESCAPE may be set for each command during 86-DOS customization. The standard escape sequences correspond to the special function keys of a VT-52 or similar terminal, as noted in each case by parentheses.

ESC S (F1) - Copy one character from the template to the new line.

ESC T (F2) - Must be followed by any character. Copies all characters from the template to the new line, up to but not including the next occurrence in the template of the specified character. If the specified character does not occur, nothing is copied to the new line.

ESC U (F3) - Copy all remaining characters in the template to the new line.

ESC V (F4) - Skip over one character in the template.

ESC W (F5) - Must be followed by any character. Skips over all characters in the template, up to but not including the next occurrence in the template of the specified character. If the specified character does not occur, no characters are skipped.

ESC P (BLUE) - Enter insert mode. As additional characters are typed, the current position in the template will not advance.

ESC Q (RED) - Exit insert mode. The position in the template is advanced for each character typed. When editing begins, this mode is selected by default.

ESC R (GRAY) - Make the new line the template. Prints an "$@", a carriage return, and a line feed. Buffer is set to empty and insert mode is turned off.

11 - Check console status. If a character is waiting at the console, AL will be FF hex on return. Otherwise, AL will be 00.
Disk I/O Functions

Disk files are identified by a disk drive code, a file name of up to 8 characters, and an extension of up to 3 characters. The drive code may explicitly specify a drive, or the default drive may be used. Case is irrelevant in the file name or extension, since only upper case is used internally. If the file name or extension includes a question mark ("?") in any position, then that position will match any character. Thus a single file name with embedded question marks may match more than one directory entry.

Generally, functions operating on disk files will use a File Control Block, or FCB. The FCB is a 33- or 36-byte segment of memory with information about a file, formatted as follows:

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 14-31 - 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 33-35 - Random Record. This 24-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 128-byte record.

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.
DISK I/O FUNCTIONS (Continued)

An unopened FCB is one in which only the first 12 bytes have been filled in, i.e., name and drive code. An opened FCB is one that has been through a successful open or create operation (Functions 15 or 22) and has its Random Record or Current Block/Current Record fields set as necessary.

13 - Disk reset. Selects drive A as the default drive, sets the disk transfer address to DS:80 hex, and flushes all file buffers. Files which have been changed in size will not be properly recorded in the disk directory until they are closed. This function need not be called before a disk change if all files which have been written to are closed.

14 - Select disk. The drive specified in DL (0=A, 1=B, etc.) is selected as the default disk. If the DL does not represent a valid drive number, then the default drive is not changed. In either case, AL returns with the number of drives.

15 - Open file. On entry, DS:DX point to an unopened FCB. The disk directory is searched for the named file and AL returns FF hex if it is not found. If it is found, AL will return a 00 and the FCB is filled as follows:

If the Drive Code was zero (default disk), it is changed to actual disk used (A=1, B=2, etc.). This allows changing the default disk without interfering with subsequent operations on the file.

The Current Block field is set to zero.

All remaining fields, up to but not including the Current Record field, are filled with system information. It is the calling program’s responsibility to set the Current Record or Random Record fields as necessary.

16 - Close file. This function must be called after file writes to insure all directory information is updated. On entry, DS:DX point to an opened FCB. The disk directory is searched and if the file is found, its position is compared with that kept in the FCB. If the file is not found in its correct position in the directory, it is assumed the disk has been changed and AL returns FF hex. Otherwise, the directory update is completed and AL returns 00.
DISK I/O FUNCTIONS (Continued)

17 - Search for first entry. On entry, DS:DX point to an unopened FCB. The disk directory is searched for the first matching name and if none is found, AL returns FF hex. Otherwise, the first 33 bytes at the current disk transfer address are filled with the directory entry and AL returns 00. The first byte is the drive number (A=1, B=2, etc.) and the next 11 bytes are the 8-character file name and 3-character extension. Note that this is the format of an unopened FCB.

18 - Search for next entry. After Function 17 has been called and found a match, Function 18 may be called to find the next match in the directory. Additional matches will be found because of duplicate names or because of "?"s appearing in the file name. Return information is the same as Function 17. DS:DX must point to the same FCB used earlier by Function 17, and this FCB must be unchanged (including no OPEN or CREATE operations on it) because it includes information necessary to continue the search.

19 - Delete file. On entry, DS:DX point to an unopened FCB. All matching directory entries are deleted. If no directory entries match, AL returns FF, otherwise AL returns 00.

20 - Sequential read. On entry, DS:DX point to an opened FCB. The 128-byte record addressed by the Current Block and Current Record fields is loaded at the disk transfer address, then the record address is incremented. If end-of-file in encountered, AL returns 01, otherwise AL returns 00.

21 - Sequential write. On entry, DS:DX point to an opened FCB. The 128-byte record addressed by the Current Block and Current Record fields is written from the disk transfer address, then the record address is incremented. If the disk is full, AL returns 01, otherwise AL returns 00.

22 - Create file. On entry, DS:DX point to an unopened FCB. The disk directory is searched for a file of the same name, or failing that, any empty entry, and AL returns FF hex if none is found or the file name is invalid (such as imbedded "?"). Otherwise, the entry is initialized to a zero-length file, the file is opened (see Function 15), and AL returns 00.
DISK I/O FUNCTIONS (Continued)

23 - Rename file. On entry, DS:DX point to a modified FCB which has a
drive code and file name in the usual position, and a second file name
starting 6 bytes after the first (DS:DX+17) in what is normally reserved
area. Every matching occurrence of the first file name is changed to the
second name. If question marks (3F hex) appear in the second file name,
then the corresponding positions in the original name will be unchanged.
AL returns FF hex if no match was found, otherwise 00.

25 - Current disk. AL returns with the code of the current default drive
(0=A, 1=B, etc.).

26 - Set disk transfer address. The disk transfer address is set to
DS:DX.

27 - Allocation table address. On return, DS:BX point to the allocation
table for the current drive, DX has the number of allocation units, and
AL has the number of records per allocation unit. This function is
intended only for system utilities written by SCP.

31 - Disk parameter address. On return, DS:BX point to an internal table
of parameters for the current default disk. This function is intended
only for system utilities written by SCP.

33 - Random read. On entry, DS:DX point to an opened FCB. The Current
Block and Current Record are set to agree with the Random Record field,
then the 128-byte record addressed by these fields is loaded at the disk
transfer address. If end-of-file is encountered, AL returns 01,
otherwise AL returns 00.

34 - Random write. On entry, DS:DX point to an opened FCB. The Current
Block and Current Record are set to agree with the Random Record field,
then the 128-byte record addressed by these fields is written from the
disk transfer address. If the disk is full, AL returns 01, otherwise AL
returns 00.

35 - File size. On entry, DS:DX point to an unopened FCB. The disk
directory is searched for the first matching entry and if none is found,
AL returns FF hex. Otherwise the Random Record field is set with the
size of the file (in 128-byte records) and AL returns 00.
DISK I/O FUNCTIONS (Continued)

36 - Set Random Record field. On entry, DS:DX point to an opened FCB. This function sets the Random Record field to the same file address as the Current Block and Current Record fields.

39 - Random block read. On entry, DS:DX point to an opened FCB, and CX contains a record count which must not be zero. The specified number of records are read from the file address specified by the Random Record field into the disk transfer address. If end-of-file is reached before all records have been read, then AL returns 01. If wrap-around above address FFFF hex in the disk transfer segment would occur, as many records as possible are read and AL returns 02. If all records are read successfully, AL returns 00. In any case, CX returns with the actual number of records read, and the Random Record and Current Block/Current Record fields are set to address the next record (the first record NOT read).

40 - Random block write. On entry, DS:DX point to an opened FCB, and CX contains a record count. The specified number of records are written from the disk transfer address to the file address specified by the Random Record field. If successful, AL returns 00. If there is insufficient space on the disk, AL returns 01, no records are written, but CX returns the maximum number of records that could be written. If wrap-around above address FFFF hex in the disk transfer segment would occur, no records are written and AL returns 02.

A special case of this function is invoked when CX=0 on entry. This causes the file size to be set to length specified by the Random Record field—upon completion, the Random Record field will point to the first record beyond the end-of-file. The file will be extended or shortened as necessary to achieve the requested length. This provides a means to pre-allocate files, or to shorten existing files. If there is insufficient disk space to extend the file as requested, then AL returns 01 and the file size is not changed. Otherwise, AL returns 0.
DISK I/O FUNCTIONS (Continued)

41 - Parse file name. On entry, DS:SI point to a command line to parse, and ES:DI point to an empty portion of memory to be filled in with an unopened FCB. If AL = 1, then leading separators are scanned off the command line at DS:SI. If AL = 0, then no scan-off of leading separators takes place.

The command line is parsed for a file name of the form x:filename.ext, and if found, a corresponding unopened FCB is created at ES:DI. If no drive specifier is present, then the default drive is assumed. If no extension is present, it is assumed to be all blanks. If the character "*" appears in the file name or extension, then all remaining characters in the file name or extension are set to "?".

If either a "?" or "*" appears in the file name or extension, then AL returns 01, otherwise 00. DS:SI will return pointing to the first character after the file name. If no valid file name was present, ES:DI+1 will point to a blank.

**Miscellaneous Functions**

0 - Program terminate. The Terminate and Ctrl-C Exit addresses are restored to the values they had on entry to the terminating program. All file buffers are flushed, but files which have been changed in length but not closed will NOT be recorded properly in the disk directory. Control transfers to the Terminate address.

37 - Set vector. The interrupt type specified in AL is set to vector to the address DS:DX. See the section on interrupt table usage for a list of certain pre-defined interrupt types.

38 - Create new program segment. On entry, DX has the segment number at which to set up a new program segment. The entire 100 hex area at location zero in the current program segment is copied into location zero of the new program segment. The memory size information at location 6 is updated, and the current Terminate and Ctrl-C Exit addresses are saved in the new program segment starting at 0A hex.
Using Operating System Functions

Disk File Reading and Writing

It is strongly recommended that all disk I/O use the block read and block write functions, Functions 39 and 40, rather than Functions 20, 21, 33, or 34. Since the block read and write functions update the Random Record field of the FCB, they may be used for sequential access as well as random, or any intermixing. Programs which would ordinarily sequentially read or write one record at a time might experience considerable improvement in performance if several records were buffered instead of just one. The block I/O functions allow this buffer size to be variable, depending, for example, on available memory size.

The Line Editor: Function 10

The most straightforward use of the editing features provided by Function 10, buffered console input, is allowing the user to correct mistakes in the line currently being entered. However, there are two other important uses, both of which take advantage of the fact that a template may already be present in the input buffer before the system call is made.

The simpler of the two is used by COMMAND and all other standard 86-DOS programs. By simply re-using the same buffer each time an input line is requested, then the previous line entered becomes the template for the new line. This allows the user to easily repeat a command, or to correct an error in the previous command. Or when used with a BASIC interpreter, for example, the user could correct the last program line entered (since the line number insures the old line will be replaced), or the line number could be changed so that several similar lines could be entered easily.

If the program wishes to actively use the editing features, it may load any arbitrary text into the buffer before requesting Function 10. Note that the second byte of the buffer must be set with the character count and an ASCII carriage return must immediately follow the text in the buffer. EDLIN, the text editor provided with 86-DOS, uses this method to provide editing within a line. A BASIC interpreter with an EDIT command could load the specified line into the buffer and let Function 10 do the rest. Any program in which there is a "typical response" at a given moment could make the template this response to allow the user to select it easily.

It is important for any program that wishes to provide line editing to use the features of Function 10 to do so. This provides the user with a set of editing operations that are consistent from program to program, and that have been tailored in one step to match the user's terminal (during 86-DOS customizing).
Running a User Program

The operating system core provides no direct means to run user programs. Instead, to run a given program represented by a disk file, the file must be opened and read into memory using the normal system functions. These functions are requested by the user program that is currently running.

The first user program to run is the initialization routine that follows a system boot, which normally loads and executes the file COMMAND.COM. This is a user program that accepts commands from the console and translates them into system function calls. COMMAND includes the capability to load and execute other program files; when these other programs terminate, COMMAND regains control. Thus COMMAND is responsible for the initial conditions that are present when a program is executed.

A standard set of initial conditions is provided by COMMAND on entry to another program. It is possible for programs other than COMMAND to load and execute program files, and they must also provide the same initial conditions so that a consistent interface may be assumed by the newly executing program. These initial conditions are as follows:

All four segment registers have the same value, and the corresponding absolute memory address is the base of a "program segment". The program is loaded and begins execution at location 100 hex in the program segment. Other assignments in the program segment are:

00 - 01: Termination point. Contains an interrupt type 20 hex, which returns control to the originating program. Thus a JMP 0 or INT 20H are the normal ways to terminate a program.

02 - 03: Memory size. Contains the first segment number after the end of memory.

05 - 05: Alternate function request entry point. See "Requesting a Function".

06 - 07: Segment size. This is the number of bytes available in the program segment.

08 - 21: Reserved.

22 - 5B: Default stack. The stack pointer is initially 5A hex, with a word of zeros on the top. Thus executing a "return" instruction will cause a transfer to location 0 and the program will terminate normally. This stack may be used as-is, or a new one may be set up. Remember that 32 bytes of stack space are required to perform system calls.

5C - 67,
6C - 77: Formatted parameters. Each of these areas may contain a
parameter, usually a file name. The first byte of each area is zero unless a disk drive is being specified, in which case 1=drive A, 2=drive B, etc. The rest is blanks if no parameter is present. No lower-case letters are allowed in these fields—they must be converted to upper case. If the parameter is a file name, then the next 8 bytes have the name, followed by the 3-character extension. Thus each parameter is properly formatted as an unopened FCB, except that the reserved area of the first overlaps onto the second. If both parameters are used as file names, the second one must be moved to a different area or it will be destroyed when the first is opened.

80 – FF: Unformatted parameters. Any information to be passed may be placed in this area. The disk transfer address is initially set to 80 hex.

COMMAND prepares the parameter areas from the console input line that specified the program to be executed. For example, if COMMAND sees an line of the form

   <programe> <file1> <file2>

this is a request to execute the file <programe>.COM. <file1> and <file2> each may or may not include a disk specifier or a file name extension, but in any case they appear in the formatted parameters at 5C hex and 6C hex. In addition, the entire input line after the last letter of <programe> appears in the unformatted parameter area beginning at 81 hex, with the number of characters placed at 80 hex.

Suppose the input line is

COPY T.BAK B:TEST.ASM

The formatted parameter at 5C hex will contain

   00 "T    BAK"

at 6C hex will be

   02 "TEST    ASM"

and at 80 hex will be

   17 "T.BAK B:TEST.ASM"

where the 17 is decimal.
Customizing 86-DOS

Setting the Special Editing Commands

The escape codes used by Function 10, buffered console input, can be set for the convenience of the user. For each special editing command, two escape codes are allowed. They are set in a table starting at address 0003 in 86-DOS. The beginning of 86-DOS looks like this:

```
0000     JMP     INIT
0003     ESCTAB:
0003     DB      "SC" ;Copy one character from template
0005     DB      "VN" ;Skip over one character in template
0007     DB      "TA" ;Copy up to specified character
0009     DB      "WS" ;Skip up to specified character
000B     DB      "UH" ;Copy rest of template
000D     DB      "HH" ;Kill line with no change in template (Ctrl-X)
000F     DB      "RH" ;Cancel line and update template
0011     DB      "DD" ;Backspace (same as Ctrl-H)
0013     DB      "PE" ;Enter Insert mode
0015     DB      "QL" ;Exit Insert mode
0017     DB      1BH,1BH ;Escape sequence to represent escape character
```

For example, the character sequences ESC S or ESC C will copy one character from the template to the new line. Note that there are three entries with the same letter for both codes. This is simply a way to make only one code available for that function.

The last entry in the table is the escape sequence to be used to pass the ESC character (1B hex). In the standard table shown here, this is done by typing ESC twice, but it could also be set up for any other escape sequence.
Customizing the I/O Section

In order to provide the user with maximum flexibility, the disk and simple device I/O handlers of 86-DOS are a separate subsystem which may be configured for virtually any real hardware. This I/O system is located starting at absolute address 400 hex, and may be any length. The DOS itself is completely relocatable and normally starts immediately after the I/O system.

Beginning at the very start of the I/O system (absolute address 400 hex) is a series of 3-byte jumps (long intra-segment jumps) to various routines within the I/O system. These jumps look like this:

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>JMP</td>
<td>INIT</td>
<td>System initialization</td>
</tr>
<tr>
<td>0003</td>
<td>JMP</td>
<td>STATUS</td>
<td>Console status check</td>
</tr>
<tr>
<td>0006</td>
<td>JMP</td>
<td>CONIN</td>
<td>Console input</td>
</tr>
<tr>
<td>0009</td>
<td>JMP</td>
<td>CONOUT</td>
<td>Console output</td>
</tr>
<tr>
<td>000C</td>
<td>JMP</td>
<td>PRINT</td>
<td>Printer output</td>
</tr>
<tr>
<td>000F</td>
<td>JMP</td>
<td>AUXIN</td>
<td>Auxiliary input</td>
</tr>
<tr>
<td>0012</td>
<td>JMP</td>
<td>AUXOUT</td>
<td>Auxiliary output</td>
</tr>
<tr>
<td>0015</td>
<td>JMP</td>
<td>READ</td>
<td>Disk read</td>
</tr>
<tr>
<td>0018</td>
<td>JMP</td>
<td>WRITE</td>
<td>Disk write</td>
</tr>
<tr>
<td>001B</td>
<td>JMP</td>
<td>FLUSH</td>
<td>Empty disk buffers</td>
</tr>
</tbody>
</table>

The first jump, to INIT, is the entry point from the system boot. All the rest are entry points for subroutines called by the DOS. Inter-segment calls are used so that the code segment is always 40 hex (corresponding to absolute address 400 hex) with a displacement of 3, 6, 9, etc. Thus each routine must make an inter-segment return when done (RET L with our assembler).

The function of each routine is as follows:

INIT - System initialization

Entry conditions are established by the system bootstrap loader and should be considered unknown. The following jobs must be performed:

A. All devices are initialized as necessary.

B. A local stack is set up and DS:SI are set to point to an initialization table. Then an inter-segment call is made to the first byte of the DOS, using a displacement of zero. For example:

- MOV AX,CS ; Get current segment
- MOV DS,AX
- MOV SS,AX
- MOV SP,STACK
- MOV SI,INITTAB
- CALL 0,DOSSEG

The initialization table provides the DOS with information about the disk system. The first byte is the number of drives (16 or fewer), followed by two 2-byte entries for each drive. The first of the two entries for each drive is the address (in the same data segment) of a disk drive parameter table (DPT). Similar drives may point to the same DPT.
CUSTOMIZING THE I/O SECTION (Continued)

Below is a brief description of each entry of the DPT. The format of the DPT will be significantly different (easier to change) in Version 1.0 of 86-DOS, and more information will be available then.

1. Number of 128-byte records per physical sector. 1 byte.

2. Number of 128-byte records per allocation unit. 1 byte.

3. Number of reserved 128-byte records at beginning of disk. 2 bytes.

4. Size of allocation table, in 128-byte records. Each allocation unit (item 7) requires 1.5 bytes in the allocation table. 1 byte.

5. Number of allocation tables kept on the drive. 1 byte.

6. Number of 128-byte records devoted to the directory. There are 8 directory entries per record. 1 byte.

7. Number of allocation units on the drive. 2 bytes.

The second of the two entries for each drive is the displacement of the allocation table for that drive. Normally, the first drive will be given a displacement of zero, and each subsequent drive will be assigned a space immediately after the previous drive's table ends. The size of the table for any drive is 128 bytes times the number of records specified in item 4 above. Note that no space need be provided by the I/O system for the allocation tables; this space is assigned by the DOS during initialization.

On the next page is a sample of an initialization table.
CUSTOMIZING THE I/O SECTION (Continued)

Below is a sample of a complete initialization table for four single-density IBM format disk drives:

<table>
<thead>
<tr>
<th>INITTAB</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>4</td>
<td>; Number of drives</td>
<td></td>
</tr>
<tr>
<td>DW</td>
<td>DRIVE0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DW</td>
<td>AT0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DW</td>
<td>DRIVE1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DW</td>
<td>AT1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DW</td>
<td>DRIVE2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DW</td>
<td>AT2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DW</td>
<td>DRIVE3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DW</td>
<td>AT3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| DRIVE0: |                |                |                |
| DRIVE1: |                |                |                |
| DRIVE2: |                |                |                |
| DRIVE3: |                |                |                |
|         | ; All drives are defined the same |                |                |
| DB      | 1              | ; Records/sector |
| DB      | 4              | ; Records/allocation unit |
| DW      | 52             | ; Reserved records (two tracks) |
| DB      | 6              | ; Allocation table size, records |
| DB      | 2              | ; Number of allocation tables (1 backup) |
| DB      | 8              | ; Number of directory records (64 entries) |
| DW      | 482            | ; Number of allocation units (512 bytes ea.) |
| ORG     | 0              | ; Allocation tables are in their own segment |

| AT0:    | DS 300H        | ; Six 128-byte records |
| AT1:    | DS 300H        |                |
| AT2:    | DS 300H        |                |
| AT3:    | DS 300H        |                |

C. When the DOS returns to the INIT routine in the I/O system, DS has the segment of the start of free memory, where a program segment has been set up. The remaining task of INIT is to load and execute a program at 100 hex in this segment, normally COMMAND.COM. The steps are:

1. Set the disk transfer address to DS:100H.

2. Open COMMAND.COM. If not on disk, report error.

3. Load COMMAND using the block read function (Function 39). If end-of-file was not reached, or if no records were read, report an error.

4. Set up the standard initial conditions and jump to 100 hex in the new program segment.
CUSTOMIZING THE I/O SECTION (Continued)

An example of code which performs this task is given:

```assembly
MOV    DX,100H
MOV    AH,26       ;Set transfer address to DS:100H
INT    21H
MOV    BX,DS      ;Save segment for later
; DS must be set to CS so we can point to the FCB
MOV    AX,CS
MOV    DS,AX
MOV    DX,FCB     ;File Control Block for COMMAND.COM
MOV    AH,15
INT    21H       ;Open COMMAND.COM
OR     AL,AL
JNZ    COMERR    ;Error if file not found
MOV    [FCB+33],0 ;Set Random Record field
MOV    B,[FCB+35],0
MOV    CX,200H    ;Load maximum records
MOV    AH,39
INT    21H       ;Block read
JCCZ   COMERR    ;Error if no records read
CMP     AL,1
JNZ    COMERR    ;Error if not end-of-file
MOV    DS,BX     ;All segment reg.s must be the same
MOV    ES,BX
MOV    SS,BX
MOV    SP,5CH     ;Stack must be 5C hex
XOR    AX,AX
PUSH    AX       ;Put zero of top of stack
MOV    DX,80H
MOV    AH,26
INT    21H       ;Set transfer address to default
PUSH    BX
MOV    AX,100H
PUSH    AX
RET     L         ;Jump to COMMAND

COMERR:
MOV    DX,BADCOM
MOV    AH,9
INT    21H       ;Print error message
STALL:  JMP    STALL ;Don’t know what to do

BADCOM:  DB     13,10,"Bad or missing Command Interpreter",13,10,"$"

FCB:    DB     1,"COMMAND.COM"
        DS     24
```

STATUS - Console input status

If a character is ready at the console, this routine returns a non-zero value in AL and the zero flag is clear. If no character is ready, AL returns zero and the zero flag is set. No registers other than AL may be changed.
CUSTOMIZING THE I/O SECTION (Continued)

CONTIN - Console input

Wait for a character from the console, then return with the character in AL. No other registers may be changed.

CONOUT - Console output

Output the character in AL to the console. No registers may be affected.

PRINT - Printer output

Output the character in AL to the printer. No registers may be affected.

AUXIN - Auxiliary input

Wait for a byte from the auxiliary input device, then return with the byte in AL. No other registers may be affected.

AUXOUT - Auxiliary output

Output the byte in AL to the auxiliary output device. No registers may be affected.

READ - Disk read
WRITE - Disk write

On entry,

\[ AL = \text{Drive number (starting with zero)} \]
\[ AH = \text{Directory flag (WRITE only)} \]
\[ CX = \text{Number of 128-byte records to transfer} \]
\[ DX = \text{Logical record number} \]
\[ DS:BX = \text{Transfer address}. \]

The number of records specified are transferred between the given drive and the transfer address. "Logical record numbers" are obtained by numbering each record sequentially starting from zero, and continuing across track boundaries. Thus for standard floppy disks, for example,
logical record 0 is track 0 sector 1, and logical record 53 is track 2 sector 2. This conversion from logical record number to track and sector is done simply by dividing by the number of records per track. The quotient is the track number, and the remainder is the record on that track. (If the first sector on a track is 1 instead of 0, as with standard floppy disks, add one to the remainder.)

"Sector mapping" is not used by this scheme, and is not recommended unless contiguous sectors cannot be read at full speed. If sector mapping is desired, however, it may be done after the logical record number is broken down into track and sector. The 8086 instruction XLAT is quite useful for this mapping.

All registers except the segment registers may be destroyed by these routines. If the transfer was successfully completed, the routines should return with the carry flag clear. If not, the carry flag should be set, and CX should have the number of records remaining to be transferred (including the record in error).

On disk writes only, register AH is zero for normal writes and non-zero for directory writes. Thus if disk I/O is being buffered in memory, as would be the case if physical sector size is greater than 128 bytes, then this memory buffer must be flushed to disk when AH is non-zero to insure the directory is updated. Version 1.0 of 86-DOS will automatically handle physical sector sizes larger than 128 bytes and buffering in the I/O area will no longer be necessary.

**FLUSH - Empty disk buffers**

This routine is called when a file is closed or when the disk system is reset. It may be used to write to disk any disk buffers that have been kept in memory. On entry, AL has the drive number whose buffers should be flushed, or if AL = -1, then flush all buffers. All registers may be destroyed except the segment registers. If memory buffering is not used, this routine may simply return (inter-segment).

Version 1.0 of 86-DOS will automatically handle physical sector sizes larger than 128 bytes and this routine will no longer be used.
Bootstrap Loader Listing

;This is a disk boot routine for the 1771/1791 type disk;
;controllers. It would normally reside on track 0,;
;sector 1, to be loaded by the "B" command of the
;monitor at address 200H. By changing the equates
;below, it may be configured to load any size of
;program at any address. The program is assumed to
;occupy consecutive sectors starting at track 0, sector
;2, and will begin execution at its load address (which
;must be a 16-byte boundary) with the Instruction
;Pointer set to zero.

; Variations are available for the Cromemco 4FDC with
; large disks, the 4FDC with small disks, the Tarbell
; single-density controller, and the Tarbell double-
; density controller. Select one.

CROMEMCOSMALL: EQU 0
CROMEMCOLARGE: EQU 0
TARBELLSSINGLE: EQU 1
TARBELLDDOUBLE: EQU 0

LOAD: EQU 400H ;Address to load program
SEG: EQU 40H ;LOAD/10H
SECTOR: EQU 51 ;No. of 128-byte sectors to load
BOOTER: EQU 200H ;"B" command puts booter here

;*****************************************************************
CROMEMCO: EQU CROMEMCOLARGE+CROMEMCOSMALL
TARBELL: EQU TARBELLSINGLE+TARBELLDDOUBLE
WD1771: EQU CROMEMCO+TARBELLSSINGLE
WD1791: EQU TARBELLDDOUBLE
SMALL: EQU CROMEMCOSMALL
LARGE: EQU CROMEMCOLARGE+TARBELL

IF SMALL
MAXSECT: EQU 18
ENDIF

IF LARGE
MAXSECT: EQU 26
ENDIF

IF TARBELL
DONEBIT: EQU 80H
DISK: EQU 78H
ENDIF

IF CROMEMCO
DONEBIT: EQU 1
DISK: EQU 30H
ENDIF
IF WD1771
READCOM: EQU 88H
ENDIF

IF WD1791
READCOM: EQU 80H
ENDIF

IF CROMEMCOLARGE
WAITBYTE: EQU 0B1H
ENDIF

IF CROMEMCOSMALL
WAITBYTE: EQU 0A1H
ENDIF

ORG BOOTER
PUT 100H

XOR AX, AX
MOV DS, AX
MOV ES, AX
MOV SS, AX
MOV SP, BOOTER ;For debugging purposes
UP
MOV DI, LOAD
MOV DX, SECTOR
MOV BL, 2

SECT:
MOV AL, 0D0H ;Force Interrupt command
OUT Disk ;To force Type I status
AAM
CMP BL, MAXSECT+1
JNZ NOSTEP
MOV AL, 58H ;Step in with update
CALL DCOM
MOV BL, 1

NOSTEP:
MOV AL, BL
OUTB Disk+2

IF CROMEMCO
MOV AL, WAITBYTE
OUT Disk+4 ;Turn on hardware wait
ENDIF

INB DISK ;Get head load status
NOT AL
AND AL, 20H
JZ OUTCOM
MOV AL, 4
BOOTSTRAP LOADER LISTING (Continued)

OUTCOM:
    OR    AL,READCOM
    OUTB  DISK
    MOV   CX,128
    PUSH  DI

READ:
    INB   DISK+4
    TEST  AL,DONEBIT

    IF    TARBELL
    JZ    DONE
    ENDF

    IF    CROMEMCO
    JNZ   DONE
    ENDF

    INB   DISK+3
    STOB
    LOOP  READ

DONE:
    POP   DI
    CALL  GETSTAT
    AND   AL,9CH
    JNZ   SECT
    ADD   DI,128
    INC   BL
    DEC   DX
    JNZ   SECT
    JMP   0,SEG

DCOM:
    OUT   DISK
    AAM

GETSTAT:
    INB   DISK+4
    TEST  AL,DONEBIT

    IF    TARBELL
    JNZ   GETSTAT
    ENDF

    IF    CROMEMCO
    JZ    GETSTAT
    ENDF

    IN    DISK
    RET
I/O Section Listing

; I/O System for 86-DOS.

; Assumes a CPU Support card at F0 hex for character I/O,
; with disk drivers for Tarbell or Cromemco controllers.

; Select disk controller here
TARBELL: EQU 1
CROMEMCO: EQU 0

; For either disk controller, a custom drive table may be defined
CUSTOM:    EQU 0

; If Tarbell disk controller, select one-sided or two-sided drives
; and single or double density controller
DOUB1SIDE: EQU 0
DOUB2SIDE: EQU 0
SNGL1SIDE: EQU 1

; If Cromemco disk controller, select drive configuration
SMALLCRO: EQU 0  ; 3 small drives
COMBCRO: EQU 0   ; 2 large drives and 1 small one
LARGECRO: EQU 0  ; 4 large drives

; Use table below to select head step speed. Step times for 5" drives is double
; that shown in the table. Times for Fast Seek mode (Cromemco controller with
; PerSci drives) is very small - 200-400 microseconds.

; Step value  1771   1791
;          0       6ms  3ms
;          1       6ms  6ms
;          2      10ms 10ms
;          3      20ms 15ms

STPSPD:    EQU 1

; Some drives require a delay between writing and stepping so that the tunnel
; erase operation does not smear across data. If needed, set ERASE to 1 and
; set WRTDLY for the amount of the delay (software loop). For a given delay in
; microseconds, WRTDLY = (delay * 8) / 18.

ERASE:  EQU 1
WRTDLY: EQU 236 ; 530 microseconds
I/O SECTION LISTING (Continued)

; Some disk drives cannot be driven at full speed (6 tracks/sec), even if the ; full head step delay is used. To slow them down, a "verify" can be performed ; after each head step during a continuous read or write operation. Select by ; setting VERIFY to VERON, disable with VEROFF.

VERON: EQU STPSPD+4
VEROFF: EQU 0
VERIFY: EQU VERON

;***********************************************************************

WD1791: EQU DOUBLSIDE+DOUBLSIDE
WD1771: EQU CROMEMCO+SNGLSIDE

IF WD1791
READCOM:EQU 80H
WRITECOM:EQU 0A0H
ENDIF

IF WD1771
READCOM:EQU 88H
WRITECOM:EQU 0A8H
ENDIF

IF TARBELL
DONEBIT:EQU 80H
DISK: EQU 78H
ENDIF

IF CROMEMCO
DONEBIT:EQU 1
DISK: EQU 30H
ENDIF

DOSSEG: EQU 80H

ORG 0
PUT 100H

BASE: EQU 0F0H
STAT: EQU BASE+7
DAV: EQU 2
TRMR: EQU 1
DATA: EQU BASE+6
PSTAT: EQU BASE+0DH
PDATA: EQU BASE+0CH

JMP INIT
JMP STATUS
JMP INP
JMP OUTP
JMP PRINT
JMP AUXIN
I/O SECTION LISTING (Continued)

JMP AUXOUT
JMP READ
JMP WRITE
JMP RETL ;Flush buffers

INIT:

MOV AX,CS ;Get current segment
MOV DS,AX
MOV SS,AX
MOV SP,STACK
MOV SI,INITTAB
CALL 0,DOSSEG
MOV DX,100H
MOV AH,26 ;Set DMA address
INT 21H
MOV BX,DS ;Save segment for later
;DS must be set to CS so we can point to the FCB
MOV AX,CS
MOV DS,AX
MOV DX,FCB ;File Control Block for COMMAND.COM
MOV AH,15
INT 21H ;Open COMMAND.COM
OR AL,AL
JNZ COMERR ;Error if file not found
MOV [FCB+33],0 ;Set 3-byte Random Record field to
MOV B,[FCB+35] ; beginning of file
MOV CX,200H ;Load maximum records
MOV AH,39 ;Block read
INT 21H
JCXZ COMERR ;Error if no records read
CMP AL,1
JNZ COMERR ;Error if not end-of-file
;Make all segment registers the same
MOV DS,BX
MOV ES,BX
MOV SS,BX
MOV SP,5CH ;Set stack to standard value
XOR AX,AX
PUSH AX
MOV DX,80H
MOV AH,26
INT 21H ;Set default transfer address (DS:0080)
PUSH BX ;Put segment on stack
MOV AX,100H
PUSH AX ;Put address to execute within segment on stack
RET L ;Jump to COMMAND

COMERR:

MOV DX,BADCOM
MOV AH,9 ;Print string
INT 21H
EI

STALL: JP STALL
I/O SECTION LISTING (Continued)

BADCOM: DB 13,10,"Bad or missing Command Interpreter",13,10,"\n"
FCB: DB 1,"COMMAND.COM"
DS 24

STATUS:
  IN STAT
  AND AL,DAV
  RET L

AUXIN:
INP:
  IN STAT
  AND AL,DAV
  JZ INP
  IN DATA
  AND AL,7FH
  RET L

AUXOUT:
OUTP:
  PUSH AX

OUTLP:
  IN STAT
  AND AL,TBMT
  JZ OUTLP
  POP AX
  OUT DATA
  RET L

PRINT:
  PUSH AX
PRINLP:
  IN PSTAT
  AND AL,TBMT
  JZ PRINLP
  POP AX
  OUT PDATA
  RET L

READ:
  CALL SEEK ;Position head
  JC ERROR
RDLP:
  PUSH CX
  CALL READSECT ;Perform sector read
  POP CX
  JC ERROR
  INC DH ;Next sector number
  ADD SI,128 ;Bump address for next sector
  LOOP RDLP ;Read each sector requested
RET L
I/O SECTION LISTING (Continued)

WRITE:
    CALL SEEK ;Position head
    JC ERROR

WRTP:
    PUSH CX
    CALL WRITESECT ;Perform sector write
    POP CX
    JC ERROR
    INC DH ;Bump sector counter
    ADD SI,128 ;Bump address
    LOOP WRTP ;Write CX sectors
    OR AL,AL
    RET L

ERROR:
    SEG CS
    MOV B,[DI],-1
    RET L

SEEK:

; Inputs:
;    AL = Drive number
;    BX = Disk transfer address in DS
;    CX = Number of sectors to transfer
;    DX = Logical record number of transfer
; Function:
;    Seeks to proper track.
; Outputs:
;    AH = Drive select byte
;    DL = Track number
;    DH = Sector number
;    SI = Disk transfer address in DS
;    DI = pointer to drive’s track counter in CS
;    CX unchanged.

    MOV SI,BX ; Save transfer address
    CBW
    MOV BX,AX ; Prepare to index on drive number
    SEG CS
    MOV AL,[BX+DRVTAB]
    OUT DISK+4 ; Select drive
    IF CROMEMCO
    OR AL,80H ;Set auto-wait bit
    ENDF
    MOV AH,AL ;Save for later
    XCHG AX,DX ;26 sectors per track
    MOV DL,26
    IF CROMEMCO
    TEST DH,10H ;Check if small disk
    JNZ BIGONE
    MOV DL,18 ;18 sectors on small disk track
I/O SECTION LISTING (Continued)

BIGONE:

ENDIF

DIV AL, DL
XCHG AX, DX
INC DH
SEG CS
MOV BL, [BX+TRKPT]
ADD BX, TRKTAB
MOV DI, BX
MOV AL, DL
SEG CS
XCHG AL, [DI]
OUT DISK+1
CMP AL, DL
JZ ONTRK
MOV BH, 3
CMP AL, -1
JNZ NOHOME

; Compute track and sector
; First sector is 1, not zero
; Get this drive's displacement into track table
; BX now points to track counter for this drive
; Exchange current track with desired track
; Inform controller chip of current track
; Seek retry count
; Head position known?
; If not, home head

TRYSK:

CALL HOME

NOHOME:

MOV AL, DL
OUT DISK+3
MOV AL, ICH+STPSPD
CALL MOVHEAD
AND AL, 98H
JZ ONTRK
DEC BH
JNZ TRYSK

STC

ONTRK:

RET

SETUP:

MOV AL, OD0H
OUT DISK
PUSH AX
AAM
POP AX

; Force Interrupt command
; so Type I status will be available
; Pause 10 microseconds

IF Cromemo
TEST AH, 10H
JNZ CHKSTP
CMP DH, 18
JA STEP

CHKSTP:

ENDIF

CMP DH, 26
JBE PUTSEC

; Check for small disk
; Only 18 sectors/track on small ones
; Check for overflow onto next track
I/O SECTION LISTING (Continued)

STEP:

INC DL
MOV DH,1
MOV AL,58H+VERIFY ;Step in with update
CALL STPHEAD
SEG CS
INC B,[DI] ;Update track counter

PUTSEC:

MOV AL,DH
OUT DISK+2

IF CROMEMCO
MOV AL,AH
OUT DISK+4 ;Turn on auto-wait
ENDIF

IN DISK ;Get head load bit
NOT AL
AND AL,20H ;Check head load status
JZ CHKDRV
MOV AL,4

CHKDRV:
; Turn on 15ms head load delay if selecting a different drive
SEG CS
CMP AH,[CURDRV]
SEG CS
MOV [CURDRV],AH
JZ RET
MOV AL,4
RET

READSECT:

CALL SETUP
MOV BL,10

RDAGN:

OR AL,READCOM
OUT DISK
MOV CX,80H
PUSH SI

RLOOP:

IN DISK+4
TEST AL,DONEBIT

IF TARBELL
JZ RDONE
ENDIF

IF CROMEMCO
JNZ RDONE
ENDIF

IN DISK+3
MOV [SI],AL
INC SI
LOOP PLOOP
I/O SECTION LISTING (Continued)

RDONE:
  POP   SI
  CALL  GETSTAT
  AND  AL, 9CH
  JZ   RET
  MOV  AL, 0
  DEC  BL
  JNZ  RDAGN
STC
RET

WRITESECT:
  CALL  SETUP
  MOV  BL, 10

WRTAGN:
  OR  AL, WRITECOM
  OUT  DISK
  MOV  CX, 80H
  PUSH  SI

WRLOOP:
  IN  DISK+4
  TEST  AL, DONEBIT

  IF  TARBELL
  JZ  WRDONE
ENDIF

  IF  CROMEMCO
  JNZ  WRDONE
ENDIF

  LODB
  OUT  DISK+3
  LOOP  WRLOOP

WRDONE:
  POP  SI
  CALL  GETSTAT
  AND  AL, OFCH
  JZ   RET
  MOV  AL, 0
  DEC  BL
  JNZ  WRTAGN
STC
RET

HOME:
  IF  CROMEMCO
  TEST  AH, 40H  ; Check seek speed bit
  JNZ  RESTORE
ENDIF

  MOV  BL, 3
I/O SECTION LISTING (Continued)

TRYHOM:
  MOV  AL,0CH+STPSPD
  CALL STPHEAD
  AND  AL,98H
  JZ   RET
  MOV  AL,58H+STPSPD ;Step in with update
  CALL DCOM
  DEC  BL
  JNZ  TRYHOM
  RET

MOVHEAD:
  IF    CROMEMCO
  TEST  AH,40H ;Check seek speed bit
  JNZ   FASTSK
  ENDF

STPHEAD:
  IF    ERASE
  PUSH  AX
  MOV   AX,WRTDLY

DLYLP:
  DEC  AX
  JNZ  DLYLP
  POP  AX
  ENDF

DCOM:
  OUT   DISK
  PUSH  AX
  AAM
        ;Delay 10 microseconds
  POP   AX

GETSTAT:
  IN    DISK+4
  TEST  AL,DONEBIT

        IF    TARBEL
        JNZ   GETSTAT
        ENDF

        IF    CROMEMCO
        JZ    GETSTAT
        ENDF

        IN    DISK
        RET

        IF    CROMEMCO

RESTORE:
  MOV   AL,0C4H ;READ ADDRESS command to keep head loaded
  OUT   DISK
  MOV   AL,77H
  OUT   4
I/O SECTION LISTING (Continued)

CHKRES:
IN 4
AND AL,40H
JZ RESDONE
IN DISK+4
TEST AL,DONEBIT
JZ CHKRES
IN DISK
JP RESTORE ;Reload head

RESDONE:
MOV AL,7FH
OUT 4
CALL GETSTAT
MOV AL,0
OUT DISK+1 ;Tell 1771 we're now on track 0
RET

FASTSK:
MOV AL,6FH
OUT 4
MOV AL,18H
CALL DCOM

SKWAIT:
IN 4
TEST AL,40H
JNZ SKWAIT
MOV AL,7FH
OUT 4
MOV AL,0
RET
ENDDIF

DS 20H

STACK:

LFAT: EQU 300H
SFAT: EQU 200H
CURDRV: DS 1

LDRIVE:
DB 1 ;Records/sector
DB 4 ;Records/cluster
DW 52 ;Reserved records
DB 6 ;FAT size (records)
DB 2 ;Number of FATs
DB 8 ;Number of directory records
DW 482 ;Number of clusters on drive
I/O SECTION LISTING (Continued)

SDRIVE:
    DB  1
    DB  2
    DW 54
    DB  4
    DB  2
    DB  8
    DW 325

    IF DOUB1SIDE
    DRVTAB: DB 0,10H,20H,30H
    TRKPT: DB 0,1,2,3
    TRKTAB: DB -1,-1,-1,-1
    ENDIF

    IF DOUB2SIDE
    DRVTAB: DB 0,40H,10H,50H
    TRKPT: DB 0,0,1,1
    TRKTAB: DB -1,-1
    ENDIF

    IF SNGL1SIDE
    DRVTAB: DB 0F2H,0E2H,0D2H,0C0H
    TRKPT: DB 0,1,2,3
    TRKTAB: DB -1,-1,-1,-1
    ENDIF

    IF TARBELL
    INITTAB: DB 4 ;Number of drives
    DW  LDRIVE
    DW  FAT0
    DW  LDRIVE
    DW  FAT1
    DW  LDRIVE
    DW  FAT2
    DW  LDRIVE
    DW  FAT3

    ORG  0
    FAT0: DS LFAT
    FAT1: DS LFAT
    FAT2: DS LFAT
    FAT3: DS LFAT
    ENDIF

; Cromemco drive select byte is derived as follows:
; Bit 7 = 0
; Bit 6 = 1 if fast seek (PerSci)
; Bit 5 = 1 (motor on)
; Bit 4 = 0 for 5", 1 for 8" drives
; Bit 3 = 1 for drive 3
; Bit 2 = 1 for drive 2
; Bit 1 = 1 for drive 1
; Bit 0 = 1 for drive 0
I/O SECTION LISTING (Continued)

IF LARGECRO
; Table for four large drives
DRVTAB: DB 71H,72H,74H,78H
TRKPT: DB 0,0,1,1
TRKRTAB: DB -1,-1
INITTAB:DB 4 ;Number of drives
  DW LDRIVE
  DW FAT0
  DW LDRIVE
  DW FAT1
  DW LDRIVE
  DW FAT2
  DW LDRIVE
  DW FAT3

ORG 0
FAT0: DS LFAT
FAT1: DS LFAT
FAT2: DS LFAT
FAT3: DS LFAT
ENDIF

IF COMBCRO
; Table for two large drives and one small one
DRVTAB: DB 71H,72H,24H
TRKPT: DB 0,0,1
TRKRTAB: DB -1,-1
INITTAB:DB 3 ;Number of drives
  DW LDRIVE
  DW FAT0
  DW LDRIVE
  DW FAT1
  DW SDRIVE
  DW FAT2

ORG 0
FAT0: DS LFAT
FAT1: DS LFAT
FAT2: DS SFAT
ENDIF
I/O SECTION LISTING (Continued)

IF SMALLCRO
; Table for 3 small drives
DRVTAB: DB 21H,22H,24H
TRKPT: DB 0,1,2
TRKTAB: DB −1,−1,−1
INITTAB:DB 3
DW SDRIVE
DW FAT0
DW SDRIVE
DW FAT1
DW SDRIVE
DW FAT2

ORG 0
FAT0: DS SFAT
FAT1: DS SFAT
FAT2: DS SFAT
ENDIF

IF CUSTOM
; Table for 2 large drives without fast seek
DRVTAB: DB 31H,32H
TRKPT: DB 0,1
TRKTAB: DB −1,−1

INITTAB:DB 2
DW LDRIVE
DW FAT0
DW LDRIVE
DW FAT1

ORG 0
FAT0: DS LFAT
FAT1: DS LFAT
ENDIF