GEM DOS 1.0, Spec Version 13

May 16,1985

DRI CONFIDENTIAL; INTERNAL USE ONLY

NOT TO BE COPIED OR GIVEN TO CUSTOMERS

Revision History

Version 8 contains corrections from the 2-28 meeting, plus some new BIOS notes, and material explaining the error reporting, base page format, and user interface.

Version 9 corrects problems with the base page format, adds BIOS function 9, and includes details about the 68K C compiler.

Version 10 adds a host of minor changes, including C bindings, function numbers, and comments.

Version 11 changes many of the system function call names as well as some of the function call descriptions.

Version 12 adds more minor changes, updates the program header information and lists the standard error numbers. System Function 0x36 (Get Drive Free Space) has changed effective with Version 0x0A00 of GEM DOS.

Version 13 lists the AS68 options, adds BIOS Functions 0A and 0B, and adds more minor changes.

Table of Contents

1. Hardware Requirements (Minimum and Expanded) 1
- 1.0.1. Minimum System 1
- 1.0.2. Expanded System 2

2. Development Environment 3
- 2.0.1. Code Development 3
- 2.0.2. Code Transport 4

3. GEM DOS Development Tools 5
- 3.1. AS68 Assembler (standard) 5
  - 3.1.1. AS68 Options 5
- 3.2. LO68, LINK68 6
- 3.3. CP/M-68K C Compiler 6
  - 3.3.1. CP68 Options 6
  - 3.3.2. C068 Options 7
  - 3.3.3. C168 Options 8
- 3.4. SID68 8
- 3.5. RELMOD 8
- 3.6. MINCE 9

4. Command File Format 11
- 4.1. Description 11
- 4.2. Relocation Information 12
  - 4.2.1. Relocation Table 13
- 4.3. Typical Program Header 14

5. CCP Commands -- Preliminary 15

6. Memory Models 17

7. Base Page Format 19

8. System Status Codes 21
- 8.1. BIOS Error Codes 21
- 8.2. BDOS Error Codes 21
- 8.3. Other Errors 22

9. System Function Calls 23
- 9.1. GEM DOS Parameter Passing Conventions 23
- 9.2. Returned Data 24
- 9.3. 00: Terminate Process (Old Form) 25
- 9.4. 01: Read Character from Standard Input 26
- 9.5. 02: Write Character to Standard Output 27
- 9.6. 03: Read Character from Standard Auxiliary Device 28
- 9.7. 04: Write Character to Standard Auxiliary Device 29
- 9.8. 05: Write Character to Standard Print Device 30
- 9.9. 06: Raw I/O to Standard Input/Output 31
- 9.10. 07: Raw Input from Standard Input 32
- 9.11. 08: Read Character from Standard Input, No Echo 33

9.12. 09: Write Null Terminated String to Standard Output 34
9.13. 0A: Read Edited String from Standard Input 35
9.14. 0B: Check Status of Standard Input 36
9.15. 0E: Set Default Drive 37
9.16. 10: Check Status of Standard Output 38
9.17. 11: Check Status of Standard Print Device 39
9.18. 12: Check Status of Standard Auxiliary Device Input 40
9.19. 13: Check Status of Standard Auxiliary Device Output 41
9.20. 19: Get Default Drive 42
9.21. 1A: Set Disk Transfer Address 43
9.22. 2A: Get Date 44
9.23. 2B: Set Date 45
9.24. 2C: Get Time 46
9.25. 2D: Set Time 47
9.26. 2F: Get Disk Transfer Address 48
9.27. 30: Get Version Number 49
9.28. 31: Terminate and Stay Resident 50
9.29. 36: Get Drive Free Space 51
9.30. 39: Create a Subdirectory 52
9.31. 3A: Delete a Subdirectory 53
9.32. 3B: Set Current Directory 54
9.33. 3C: Create a File 55
9.34. 3D. Open File 56
9.35. 3E: Close File 57
9.36. 3F: Read File 58
9.37. 40: Write File 59
9.38. 41: Delete File 60
9.39. 42: Seek File Pointer 61
9.40. 43: Get/Set File Attributes 62
9.41. 45. Duplicate File Handle 63
9.42. 46: Force File Handle 64
9.43. 47: Get Current Directory 65
9.44. 48: Allocate Memory 66
9.45. 49. Free Allocated Memory 67
9.46. 4A: Shrink Size of Allocated Memory 68
9.47. 4B: Load or Execute a Process 69
9.48. 4C: Terminate Process 71
9.49. 4E: Search for First Occurance of Filespec 72
9.50. 4F: Search for Next Occurrence of Filespec 74
9.51. 56: Rename a File 75
9.52. 57: Get/Set File Date & Time Stamp 76
10. BIOS Function Calls **79**
10.1. Parameter Format In Memory 79
10.2. BIOS Initialization Sequence 79
10.3. **00** 80

10.4. **01** 81
10.5. **02** 82
10.6. **03** 83
10.7. **04** 84
10.8. **05** 85
10.9. **06** 87
10.10. **07** 88
10.11. **08** 89
10.12. **09** 90
10.13. **0A** 91
10.14. **0B** 92
11. The GEM-GSX Graphics Interface **93**

1. Hardware Requirements (Minimum and Expanded)

1.0.1. Minimum System

The GEM DOS operating system requires a minimum system configuration that supports its capabilities. In particular, this includes both RAM and ROM storage, a bit-mapped video controller and display, and a mouse or other pointing device.

- 68000 microprocessor
- Bit mapped video controller card and a display with 320 by 200 resolution
- 128 Kbytes RAM
- 160 Kbytes ROM
- An optical or track mouse, or other pointing device

1.0.2. Expanded System

While GEM DOS runs and "exercises" its features with the system configuration above, the expanded high-performance system described below uses the full capabilities of GEM DOS.

- 68010, 68020, or 68070 advanced microprocessor
- High resolution, color bit-map display and controller (640 by 400 pixel resolution, barrel shifter hardware for very high speed screen refresh)
- 256 Kbytes of RAM to support sophisticated applications and multitasking
- 192 Kbytes of ROM, including additional desk accessories
- Mouse
- Disk drives and controllers
- Optical drives and controllers
- Printer and communication ports

2. Development Environment

2.0.1. Code Development

The OEM hardware-specific code development is supported on a Motorola VME-10 system with a graphics display monitor, 256 Kbytes of additional RAM, a terminal, a VME-400 dual serial port, and a Mouse Systems serial mouse.

You have the option of coding in a cross-development environment, under CP/M-68K, or coding in GemDos 1.0 native mode. In native mode, you can use prototypes of new hardware added to the VME-10 as expansion boards.

The Apple Lisa will support an alternative native mode development environment.

2.0.2. Code Transport

You can transport GEM DOS to the target system with the "Kermit" communications program.

3. GEM DOS Development Tools

3.1. AS68 Assembler (standard)

The 68000 assembler processes an assembly language source module, producing a relocatable object module, and an optional listing.

The command line has the following format:

AS68 [-a] [-i] [-p] [-u] [-n] [-l] [-t] [-f path] [-s path] source [->list]

3.1.1. AS68 Options

The option flags are as follows:

1. "-a", if present, specifies that only short (16-bit) addresses are to be generated. This option is not currently supported by the CEM DOS relocatable load format, but could be used when generating a module to be loaded as part of the OS that will reside in the first 32K of memory.
2. "-i", if present, specifies that the assembler initialization phase is to be performed. When this option is specified, it should be the only option on the command line. The assembler will read the file AS68INIT, and produce the file AS68SYMB.DAT.
3. "-p", if present, specifies that an assembly listing is to be produced. Default listing takes place to STDOUT If a listing file is desired, or the output is to be sent to the printer, STDOUT should be redirected on the command line.
4. "-u", if present, specifies that all underlined labels are to be treated as externally defined references.
5. "-n", if present, specifies that no branch optimization is to be done by the assembler.
6. "-l", if present, specifies that only long (32-bit) addresses are to be generated.
7. "-t", if present, specifies that 68010 code is to be generated.
8. "-f path" allows the user to direct where temporary files are to be created. This would allow a hard disk or a RAM-disk to be used to improve performance.
9. "-s path" specifies the path name in which the AS68SYMB.DAT file is to be found. This allows 1 copy to be used from any path on the system.

The source file name must be spelled out in full; no default extension is supplied by the assembler.

Note: Some of these options have not been extensively tested on GEM DOS, are not fully documented, and thus are not officially supported.

3.2. LO68, LINK68

L068 is the Digital Research 68000 linker. L068 combines object modules into executable programs, resolving external references if any exist.

LINK68 is a Digital Research 68000 linkage editor that combines assembled or compiled object modules with other object modules called from an appropriate runtime library.

3.3. CP/M-68K C Compiler

The GEM DOS development environment supports the Alcyon C compiler for the 68000.

This section describes the options present on the individual passes of the CP/M-68K C compiler. Note that these options have not been extensively tested on CP/M-68K, are not documented, and thus are not officially supported.

3.3.1. CP68 Options

The preprocessor, CP68, has a number of options which control the format of preprocessor output. The command line has the following format:

CP68 [-c] [-p] [-e] [-dsymbol[=value]] [-i d:] source dest

The option flags are.

1. "-c", if present, specifies that comments are to be left in the preprocessed output. Note that a file processed in this manner must be stripped of the comments before passing the file through C168, which expects to have comments removed.

2. "-p", if present, specifies that the file and line number information which is normally included at the beginning of each line be omitted. Use this flag when preprocessing assembler files with CP68.

3. "-e", if present, specifies that the preprocessor output go to the standard output rather than the destination file.

4. "-dsymbol[=value]" provides a means of defining a preprocessor symbol on the command line. The optional "=value" string allows

assigning a value to the symbol. For instance, the construct:

```
CP68 -dx=y
```

behaves as if the following preprocessor statement were included in the source file:

```
#define X Y
```

Note that the variable and the value are both converted to upper case.

5. "-i d:" allows using a different drive / user number combination for files in a "#include <file.h>" statement. Files in an include statement with double quotes are unaffected by this option.

6. The Source and destination files are specified by "source", and "dest", respectively.

The compiler also accepts the switches "-3", "-4", "-5", "-6", and "-7". These have no function on CP/M-68K.

3.3.2. C068 Options

The Parser, C068, has a command line of the following form:

```
c068 source link icode strings [-e] [-f] [-w] [-t]
```

The source for the parser is normally the output of the preprocessor, CP68. The "link", and "icode" files are temporary files for use by the code generator, C168. The "strings" file is a temporary file for use by C068. This file is deleted by C068 after use.

The Parser options are:

1. "-e" specifies that IEEE format floating point is to be used.

2. "-f" specifies that FFP (Fast Floating Point) format floating point is to be used. Only one of the "-e" and "-f" options may be specified.

3. "-w" if specified, causes warning messages to be suppressed.

4. "-t", if specified, causes the compiler to emit code for the 68010, as opposed to the 68000. This switch is normally not required, as CP/M-68K patches MOVE from SR instructions in the user program to MOVE from CCR. "-t" is thus required only for programs which will be run standalone or will be placed in ROM.

3.3.3. C168 Options

The code generator command line has the following format:

C168 link icode asm [-[t][a][f][d]]

The "link" and "icode" files are the output of C068. The "asm" file is the generated code for input into the assembler. Options for C168 are:

1. "t" Generate 68010 code. See the note on the "-t" switch for C068. If "-t" is specified for C068, then this switch must be present for C168.

2. "a" Generate 16-bit offsets The object program must fit in the first 32K of memory to use this switch.

3. "f" This switch is accepted but has no affect under CP/M-68K.

4. "d" Include line numbers from the source file as comments in the generated assembly code

Note: the "usage" message generated by C168 in version 1.2 is incorrect.

3.4. SID68

SID68 is the Symbolic Interactive Debugger, which lets you view sections of code and disassemble them.

3.5. RELMOD

RELMOD is a Digital Research product that adapts relocatable object modules from one format to another. The version supplied with GEM DOS converts CP/M-68K relocatable object modules to GEM DOS format relocatable object modules, and also performs

conversions from GEM DOS format to CP/M-68K format.

3.6. MINCE

The MINCE program editor is supplied with the GEM DOS tools package.

4. Command File Format

4.1. Description

The GEM DOS Command file format is the output of a format conversion process using RELMOD. RELMOD is a conversion utility that takes CP/M-68K relocatable output files generated by LO68 and translates them to GEM DOS relocatable files. Only commands with contiguous text, data, and block storage segments are supported. Sixteen bit relocatable objects are not supported.

The GEM DOS Command file format begins with the same file header as described in Section 3 of the CP/M-68K Programmer's Guide. A 14-word header (601AH-type) denotes a file that contains contiguous segments.

The program segments are also the same as in CP/M-68K. The order of occurrence is text and data
followed by an optional symbol table and relocation information, if required.

4.2. Relocation Information

The offset from the beginning of the text segment to the first longword needing relocation is stored as a longword.
The relocation information stores the offset from one longword to be relocated to the next such longword. When an offset of zero is encountered, no further relocation information is to be found. Bytes are then used to store succeeding offsets. If the offset is greater than 254, the relocation information will contain a byte of 1. Then 254 will be subtracted from the desired offset and the procedure repeated.

4.2.1. Relocation Table

Relocation Offset Byte Table

Byte	Description
0	No further relocation information
1	Add 254 to the existing offset, go to the next byte
2,3	Reserved
4-254	Represents the offset to the next longword needing relocation*
255	Reserved

* plus 254 x n where n is the number of "1" bytes that preceded this byte.

All relocation information is relative to the beginning load address (i.e. start of text).

4.3. Typical Program Header

Typical GEMDOS.PRG Header for Contiguous Program Segments

Byte	Values	Size	Contents
0H	601AH	1 Word	Denotes contiguous text, data, and block storage segments
2H	2376H	1 Longword	Number of bytes in text segment
6H	422H	1 Longword	Number of bytes in data segment
0AH	1806H	1 Longword	Number of bytes in block storage segment
0EH	142H	1 Longword	Number of bytes in symbol table
12H	000H	1 Longword	Reserved, always zero
16H	000H	1 Longword	Reserved, always zero
1AH	C0H	1 Word	Reserved, always zero
		2376H Bytes	Text segment image
		422H Bytes	Data segment image
		142H Bytes	Symbol table
	X	1 Longword	Offset of first longword to relocate
	X	n Bytes	Successive offsets to further longwords to relocate
	00H	1 Byte	No more relocation information

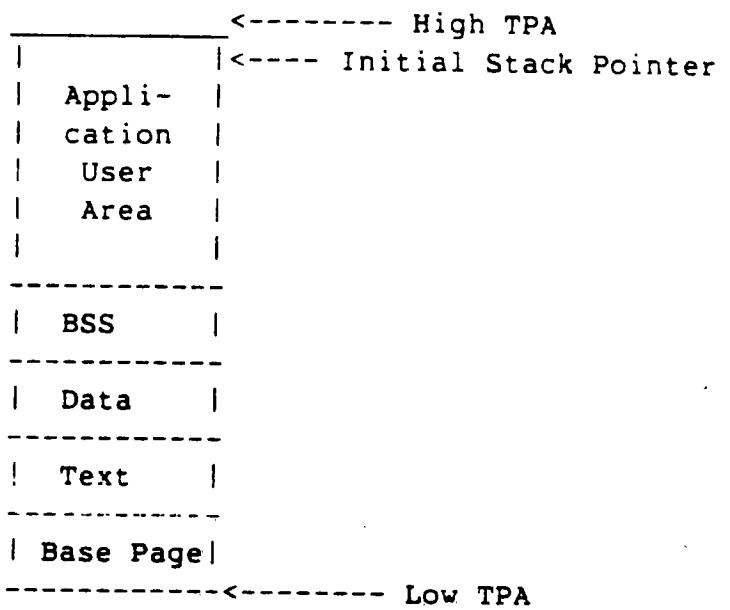
5. CCP Commands -- Preliminary

The following built-in commands are supplied with GEM DOS:

IR	List directory
TYPE	List file
CD	Change directory
MD	Make directory
RD	Remove Directory
DEL(ERA)	Delete(Erase) a file
REN	Rename a file
SHOW	Display disk statistics
COPY	Copy files

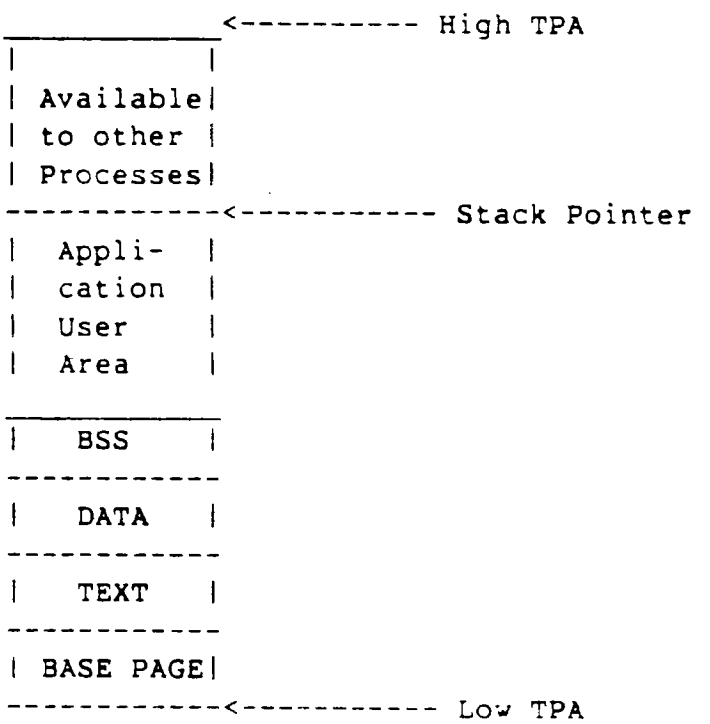
6. Memory Models

GEM DOS establishes the Transient Program Area(TPA) for an application program as follows:



The initial stack pointer is directed at the top end of the TPA. The base page contains pointers as described in Section 7. To insure maximum compatibility with future releases of GEM DOS, it is recommended that memory that is not required by the application program be freed for use by other processes. The top of the stack should be moved down to an acceptable value, and an M_SHRINK function should be performed (0x4A). If additional memory is needed later, an M_FREE function should be performed to determine the amount of available memory, followed by an M_ALLOC function (0x48) to have a portion of that memory allocated.

The TPA will now look like this:



7. Base Page Format

Offset into base page	Value	Description
00	p_lowtpa	base address of the TPA
04	p_hitpa	points to the end of TPA + 1
08	p_tbase	base address of text (code)
0C	p_tlen	length of text (code)
10	p_dbase	base address of initialized data
14	p_dlen	length of data
18	p_bbase	base address of BSS uninitialized data
1C	p_blen	length of BSS uninitialized data
2C	p_env	pointer to the environment string
80	p_cmdlin	command line image

8. System Status Codes

GEM DOS uses a simple convention to return system status and error codes. Both types of codes are returned in register D0.L.

For all functions that return some non-address value, a negative number indicates an error return. Any function that returns an address will typically return a 0 or -1 to indicate an error.

8.1. BIOS Error Codes

E_OK	OK (No error)	0L
ERROR	Fundamental error	-1L
EDRVNR	Drive not ready	-2L
EUNCMD	Unknown command	-3L
E_CRC	CRC error	-4L
EBADRQ	Bad request	-5L
E_SEEK	Seek error	-6L
EMEDIA	Unknown media	-7L
ESECNF	Sector not found	-8L
EPAPER	No paper	-9L
EWRITF	Write fault	-10L
EREADF	Read fault	-11L
EGENRL	General error	-12L
EWRPRO	Write protect	-13L
E_CHNG	Media change	-14L
EUNDEV	Unknown device	-15L
EBADSF	Bad sectors on format	-16L
EOTHER	Insert other disk	-17L

8.2. BDOS Error Codes

		GEM DOS	PC DOS Equiv.
EINVFN	Invalid function number	-32L	1
EFILNF	File not found	-33L	2
EPTHNF	Path not found	-34L	3
ENHNDL	No handles left (too many open files)	-35L	4
EACCDN	Access denied	-36L	5
EIHNDL	Invalid handle	-37L	6
ENSMEM	Insufficient memory	-39L	8
EIMBA	Invalid memory block address	-40L	9

EDRIVE	Invalid drive specified	-46L	15
ENMFIL	No more files	-49L	18

8.3. Other Errors

ERANGE	Range error	-64L
EINTRN	Internal error	-65L
EPLFMT	Invalid program load format	-66L
EGSBF	Setblock failure due to growth restrictions	-67L

9. System Function Calls

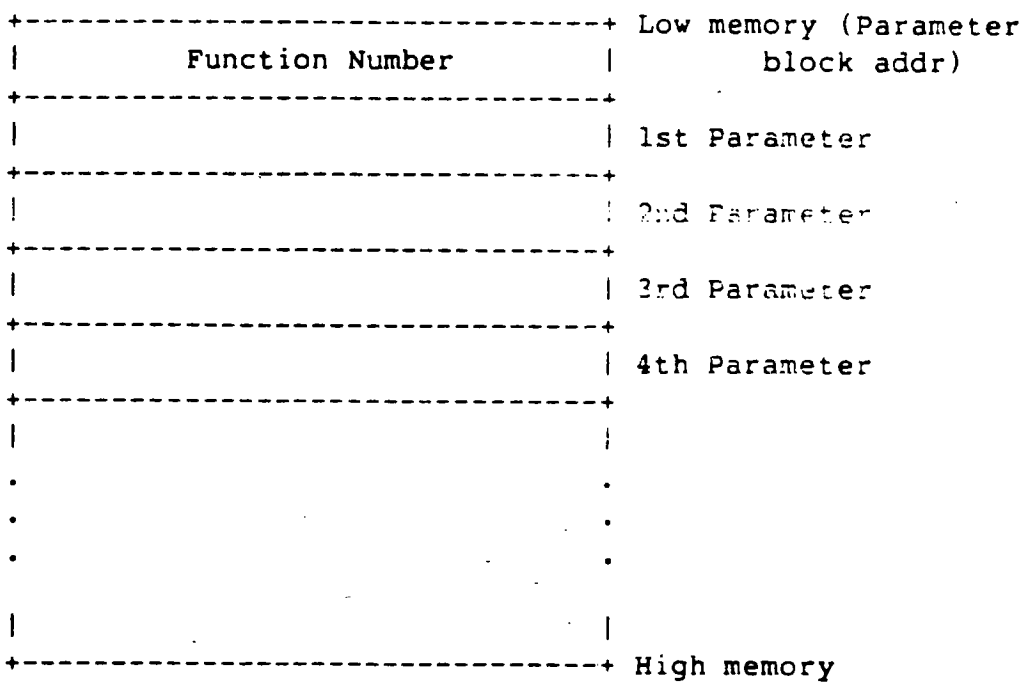
9.1. GEM DOS Parameter Passing Conventions

The GEM DOS system function calls use a common parameter passing convention. This convention supports use of system function calls by C and other high level languages.

Each system function call reference from a high level language fills a parameter block with the number of the function call, necessary parameters, and other data relevant to that function (such as drive number, time-of-day code, etc.)

Once the parameter block is built, simply perform a TRAP number1, and the operating system interprets the call and carries out the action.

The GEM DOS parameter blocks have the following form.



In this diagram, the position of the parameters and the function number correspond to the way C places them on the stack. Specifically, the C code:

CALL (functionnumber, P1, P2, P3, ... Pn)

loads the Nth parameter into the parameter block first, followed by the N-1th, and continues in this manner until the function number is entered last in the parameter block.

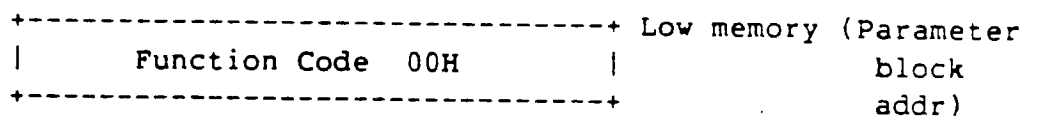
9.2. Returned Data

The default convention returns all byte, word, and long data in D0. In individual cases where more information is returned, D0 contains the address of an Information Return Block, which receives the data. The descriptions of individual function calls explain how data returns to the application.

9.3. 00: Terminate Process (Old Form)

```
   VOID
p_term0 ()
{
}
```

Parameter Block



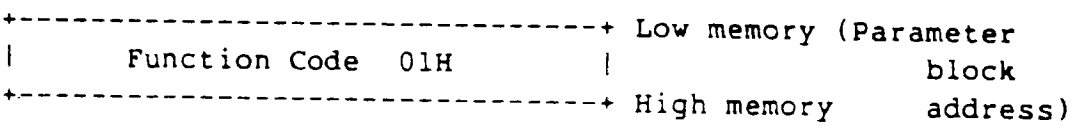
Function:

Terminate this process, return to parent process with return code set to 0.

9.4. 01: Read Character from Standard Input

```
   LONG
c_conin ()
{
}
```

Parameter Block



Return Parameters:

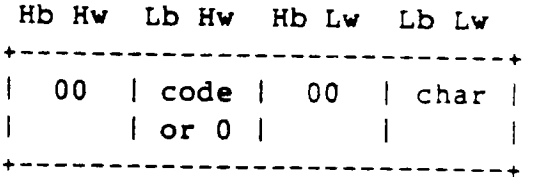
Register D0 L : Character read

Function:

Reads character from standard input and echoes it to the standard output device.

If the console is the standard input device, and if the console is GSX 2 0 compatible, the console scan code is found in the low byte of the high word, as shown below. This byte is set to 00 otherwise.

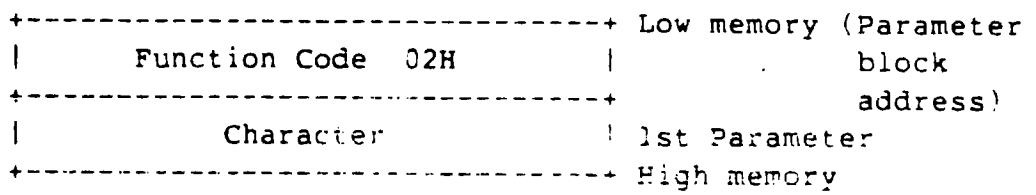
D0 Contents



9.5. 02: Write Character to Standard Output

```
   VOID
c_conout (c)
   WORD     c;
{
}
```

Parameter Block



Function:

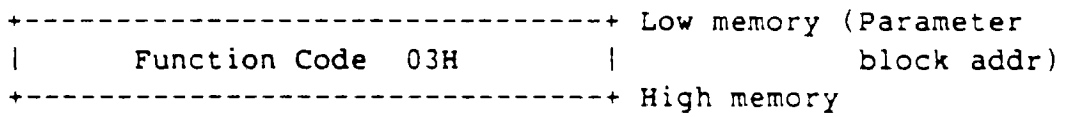
The first (and only) parameter is a word; lower 8 bits contain a character to be printed. The upper 8 bits should be zero to ensure compatibility with future extensions to the 16-bit character set.

The function displays this character on the standard output device.

9.6. 03: Read Character from Standard Auxiliary Device

```
    LONG
c_auxin ()
{
}
```

Parameter Block



Return Parameters:

Register D0.L : Character read from auxiliary port

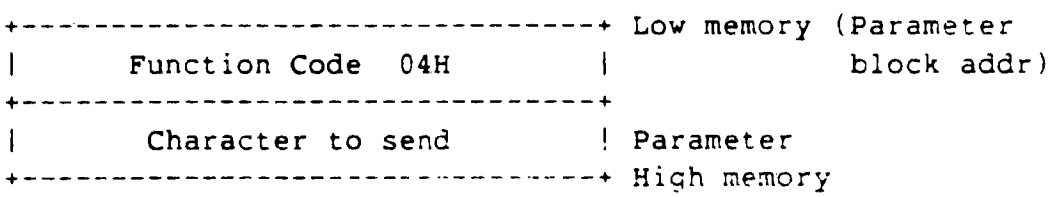
Function:

Receives character from auxiliary port and returns it in D0.L.

9.7. 04: Write Character to Standard Auxiliary Device

```
   VOID
c_auxout (c)
  WORD c;
{
}
```

Parameter Block



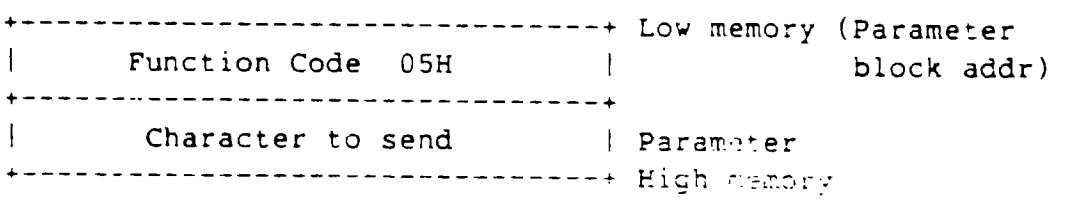
Function:

The character in the parameter block is sent to the auxiliary port. The upper 8 bits should be zero to ensure compatibility with future extensions to the 16-bit character set.

9.8. 05: Write Character to Standard Print Device

```
    VOID
c_prnout (c)
   WORD c;
{
}
```

Parameter Block



Register DOW . Character to send to printer.

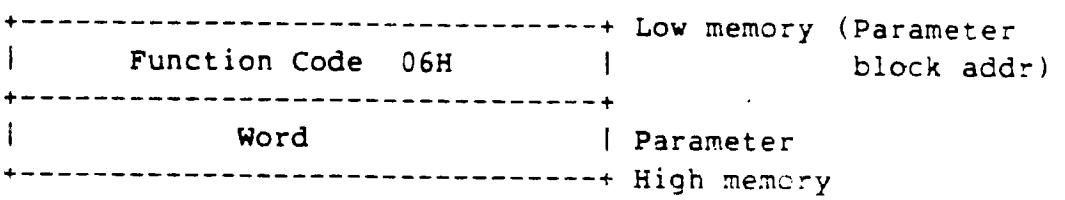
Function:

Send the specified character to the printer device. The upper 8 bits should be zero to ensure compatibility with future extensions to the 16-bit character set

9.9. 06: Raw I/O to Standard Input/Output

```
    LONG
c_rawio (parm)
    WORD    parm;
{
}
```

Parameter Block



Function:

If word = FF, read character from standard input device and return in D0.L.

If word does not equal FF, it is assumed to be a character and is sent to the standard output device.

Return Parameters:

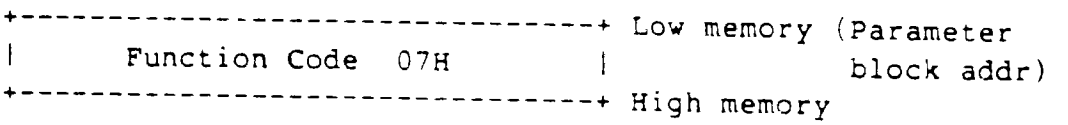
Register D0.W : If word = FF, D0.W contains character read from standard input device

If no character is available, return D0.L equals 0L.

9.10. 07: Raw Input from Standard Input

```
    LONG
c_rawcin ()
{
}
```

Parameter Block



Return Parameters:

Register D0.L : Character read from standard input device

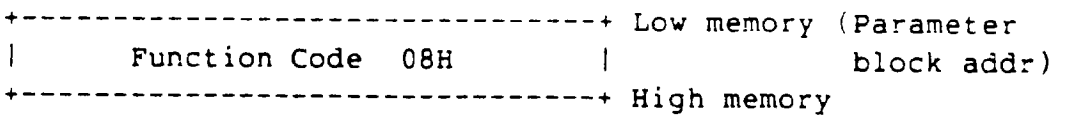
Function:

Read character from standard input device without echoing it to standard output device. Control characters pass through without trapping by the console routine.

9.11. 08: Read Character from Standard Input, No Echo

```
    LONG
c_necin ()
{
}
```

Parameter Block



Return Parameters:

Register D0.L : Character read from standard input device.

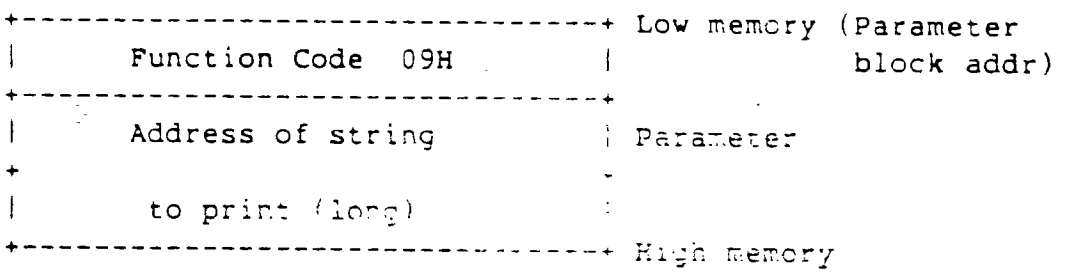
Function:

Read character from standard input device without echoing it to standard output device. Control characters (^C, ^S, and ^Q) are interpreted and have their proper effect.

9.12. 09: Write Null Terminated String to Standard Output

```
    VOID
c_conws (p)
    BYTE    *p;
{
}
```

Parameter Block



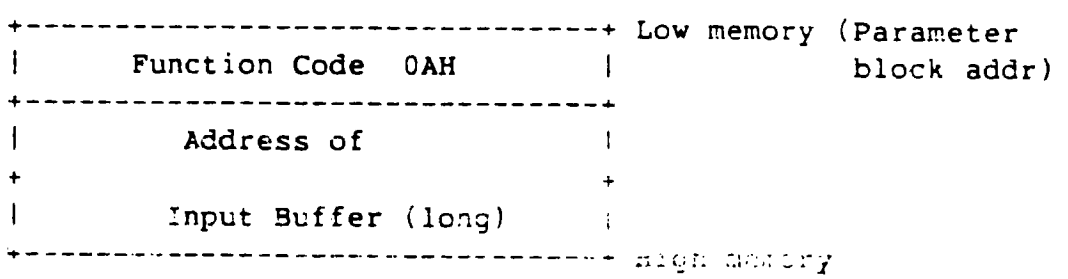
Function:

The target string is transmitted to the standard output device, character by character. A null character terminates the string.

9.13. 0A: Read Edited String from Standard Input

```
    VOID
c_conrs ()
    BYTE    *p;
{
}
```

Parameter Block



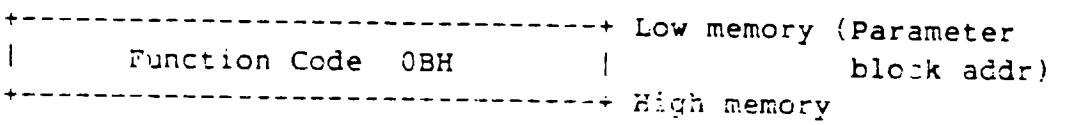
Function:

On entry, the first byte of the buffer should be set to the length of the data portion of the buffer. On return, the second byte is set to the actual length read, and the third through n bytes contain the characters read. The string is null terminated.

9.14. 0B: Check Status of Standard Input

```
    LONG
c_conis ()
{
}
```

Parameter Block



Return Parameters:

Register D0 L · Contains −1 if a character is available.
0 if no character available.

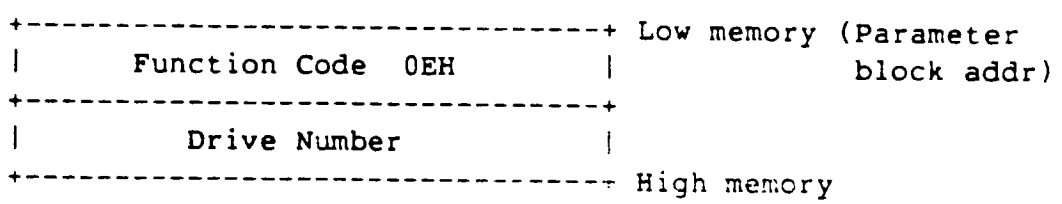
Function:

Returns the status of standard input, checking for characters to receive from it.

9.15. 0E: Set Default Drive

```
    LONG
d_setdrv (newdrv)
    WORD      newdrv;
{
}
```

Parameter Block



Drive Number:

0 = Drive A, 1 = Drive B ..., 15 = Drive P

Return Parameters:

Register D0.L : Bit map of drives in system (bit 0 = A, bit 1 = B, ...).

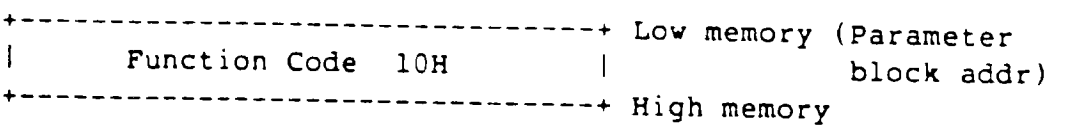
Function:

Makes a specified drive in the range A-P the current drive and returns the drive map of the system.

9.16. 10: Check Status of Standard Output

```
    LONG
c_conos ()
{
}
```

Parameter Block



Return Parameters:

Register D0.L : Contains -1 if the console is ready to receive a character;
0 if it is unavailable.

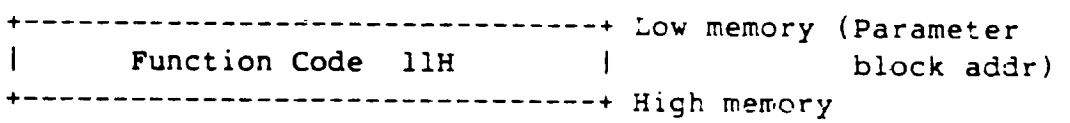
Function:

Returns the status of the console device, checking to see if it ready to receive characters.

9.17. 11: Check Status of Standard Print Device

```
    LONG
c_prnos ()
{
}
```

Parameter Block



Return Parameters:

Register D0.L : Contains -1 if the printer is ready to receive a character
0 if it is unavailable.

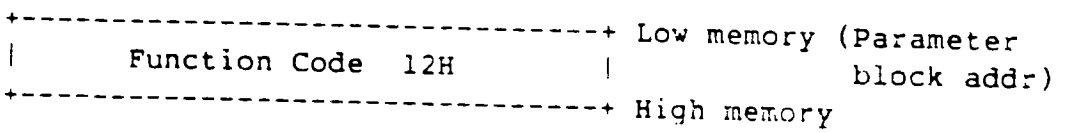
Function:

Returns the status of the printer device, checking to see if it ready to receive characters.

9.18. 12: Check Status of Standard Auxiliary Device Input

```
    LONG
c_auxis ()
{
}
```

Parameter Block



Return Parameters:

Register D0.L : Contains -1 if a character is available;
0 if no character available.

Function:

Returns the status of the auxiliary input device, checking for characters to receive from it.

9.19. 13: Check Status of Standard Auxiliary Device Output

```
    LONG
c_auxos ()
{
}
```

Parameter Block

Return Parameters:

Register D0.L . Contains -1 if the device is ready to receive a character,
0 if it is unavailable.

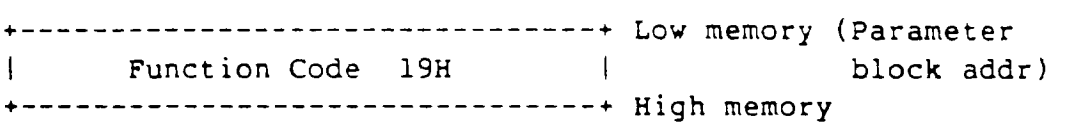
Function:

Returns the status of the auxiliary device, checking to see if it ready to receive characters.

9.20. 19: Get Default Drive

```
  LONG
d_getdrv ()
{
}
```

Parameter Block

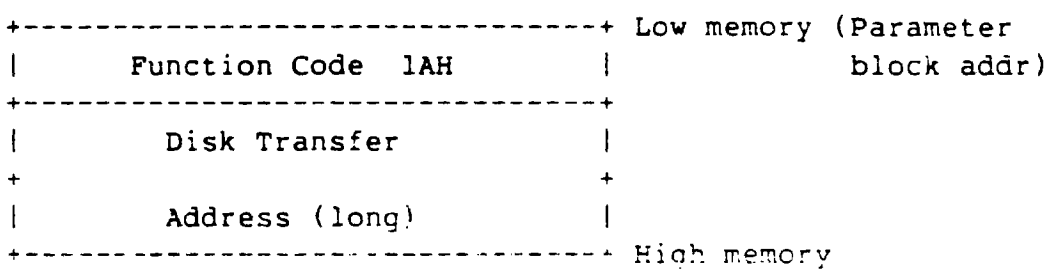


Register D0.W : Contains code of the current drive number
0 = A, 1 = B, up to 15 = P.

9.21. 1A: Set Disk Transfer Address

```
  VOID
f_setdta (b)
BYTE *b;
{
}
```

Parameter Block

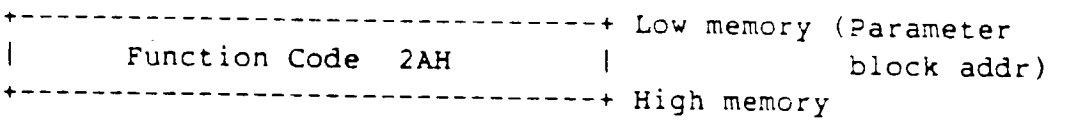


Set disk transfer address used by f_sfirst().

9.22. 2A: Get Date

```
  WORD
t_getdate ()
{
}
```

Parameter Block



Return Parameters:

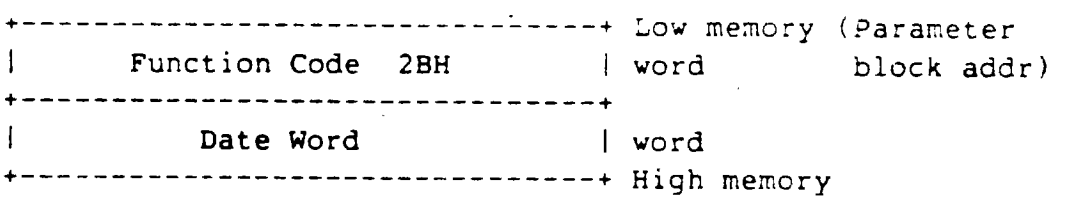
Register D0.W : Contains date in the format

Bits 0 through 4 indicate the date in the range 1 - 31
Bits 5 through 8 indicate the month in the range 1 - 12
Bits 9 through 15 indicate the year (since 1980) in the range 0-119

9.23. 2B: Set Date

```
  LONG
t_setdate (date)
WORD date;
{
}
```

Parameter Block



Date Word contains the new date in the format:

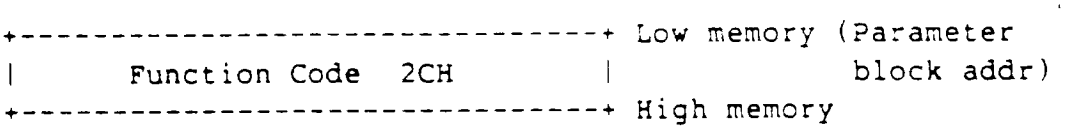
Bits 0 through 4 indicate the date in the range 1 - 31
Bits 5 through 8 indicate the month in the range 1 - 12
Bits 9 through 15 indicate the year (since 1980) in the range 0-119

Returns ERROR if the date is not valid.

9.24. 2C: Get Time

```
 WORD
t_gettime ()
{
}
```

Parameter Block



Return Parameters:

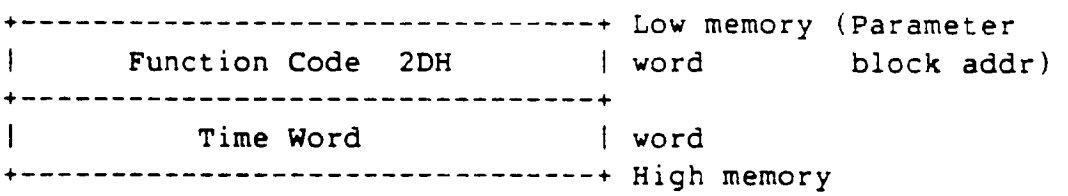
Register D0.W contains the time-of-day in the format:

Bits 0 through 4 indicate the binary number of two-second increments
Bits 5 through 10 indicate the binary number of minutes
Bits 11 through 15 indicate the binary number of hours

9.25. 2D: Set Time

```
  LONG
t_settime (time)
WORD time;
{
}
```

Parameter Block



Time Word contains the new time in the format:

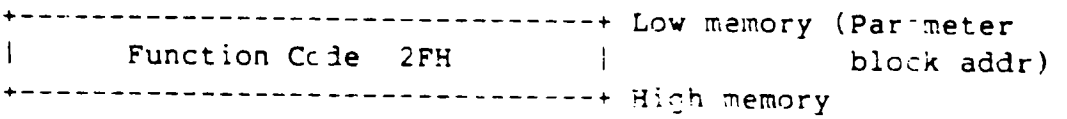
Bits 0 through 4 indicate the binary number of two-second increments
Bits 5 through 10 indicate the binary number of minutes
Bits 11 through 15 indicate the binary number of hours

Returns ERROR if the time is invalid.

9.26. 2F: Get Disk Transfer Address

```
   LONG
f_getdta ()
{
}
```

Parameter Block



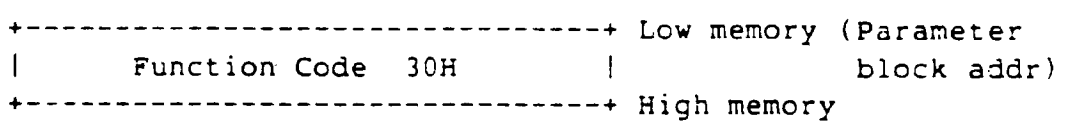
Return Parameters:

Register D0 L : Contains the current Disk Transfer Address

9.27. 30: Get Version Number

```
 WORD
s_version ()
{
}
```

Parameter Block



Return Parameters:

Register D0.W : Contains version number. For first release, this number will be 0001H.

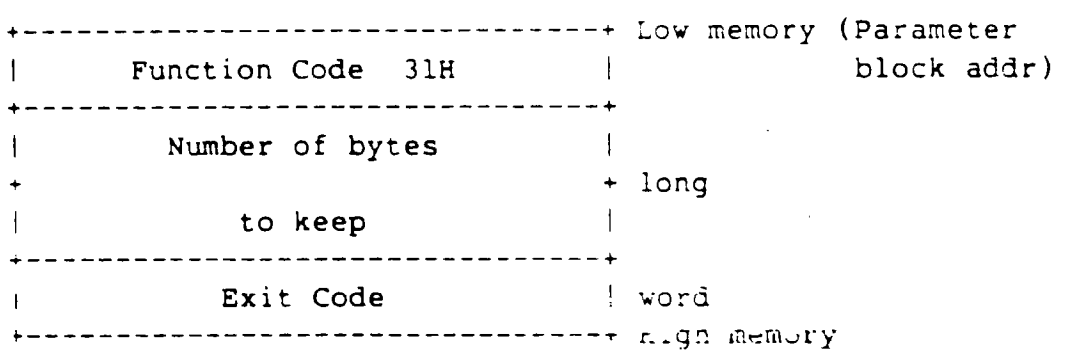
Function:

Lower byte contains major version number; upper byte contains minor version number. 0001 = version 1.00.

9.28. 31: Terminate and Stay Resident

```
 VOID
p_termres (n_bytes,rc)
LONG n_bytes
WORD rc;
{
}
```

Parameter Block



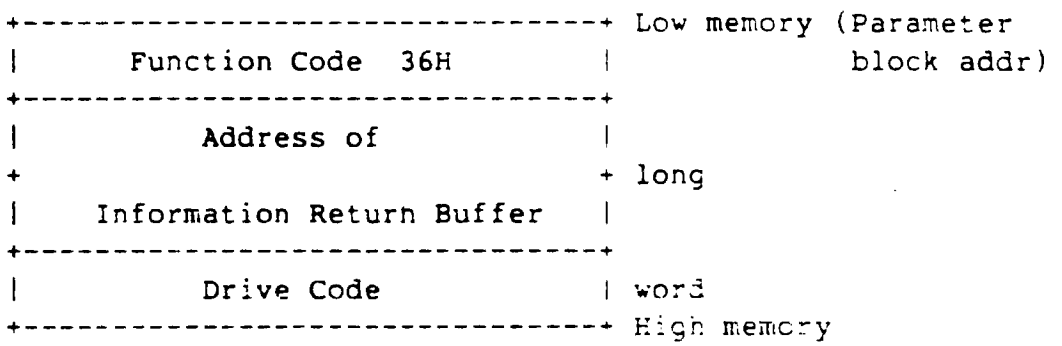
Long: Contains number of bytes to keep
Word: Contains exit code

Note: Use of this function may make an application difficult to port to a future operating systems.

9.29. 36: Get Drive Free Space

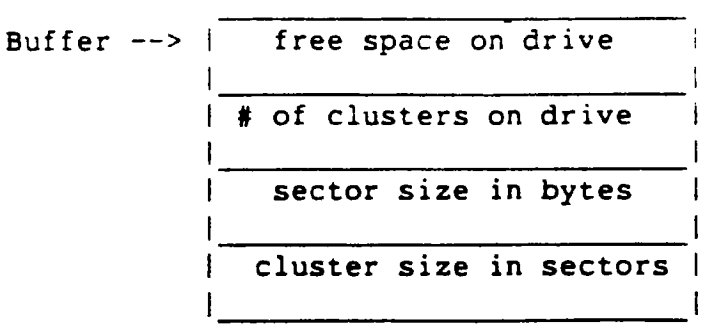
```
VOID
d_free (dr)
WORD dr;
{
}
```

Parameter Block



Drive Code: 0 = Default, 1 = A, 2 = B, etc.

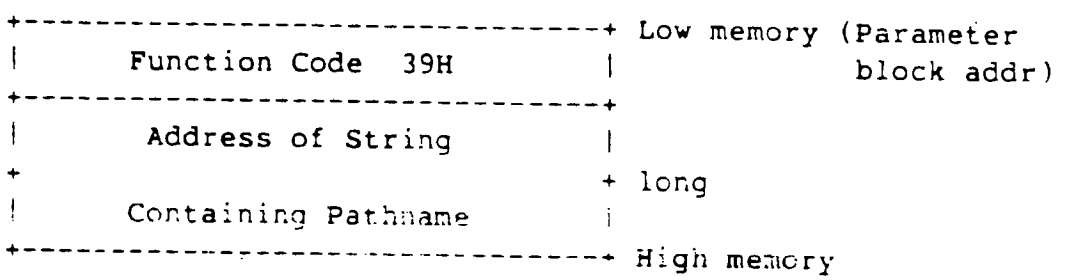
The Information Return Buffer holds the data retrived by this function call.



9.30. 39: Create a Subdirectory

```
    LONG
d_create (path),
    BYTE   *path;
{
}
```

Parameter Block



The pathname for the new subdirectory is contained in a null-terminated string. The parameter block contains the address of this string.

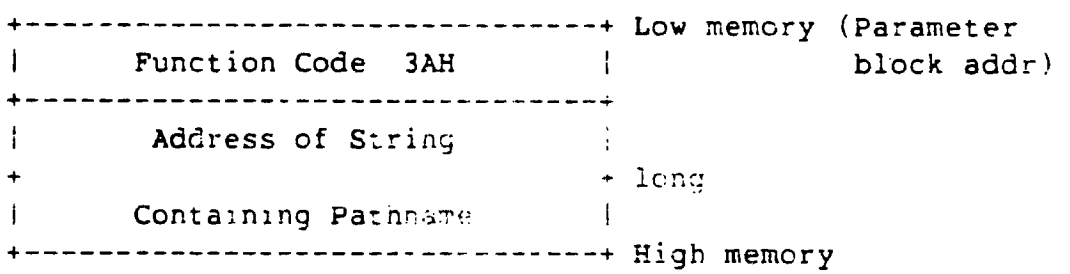
Return Parameters:

Register D0.L : Contains 0 if operation succeeds,
non-zero if error occurs.

9.31. 3A: Delete a Subdirectory

```
    LONG
d_delete (path)
    BYTE    *path;
{
}
```

Parameter Block



The pathname for the subdirectory to remove is contained in a null-terminated string. The parameter block contains the address of this string.

Return Parameters:

Register D0.L : Contains 0 if operation succeeds,
non-zero if error occurs.

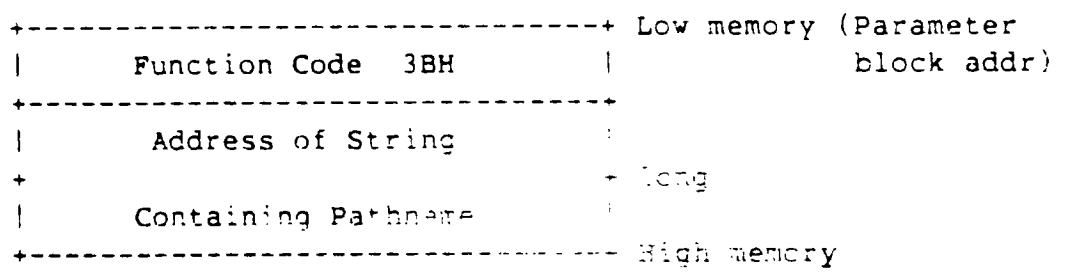
Function:

Removes a subdirectory. If the subdirectory is not empty, an error code is returned in register D0.L.

9.32. 3B: Set Current Directory

```
    LONG
d_setpath (path)
    BYTE      *path;
{
}
```

Parameter Block



The pathname for the new current directory is contained in a null-terminated string. The parameter block contains the address of this string.

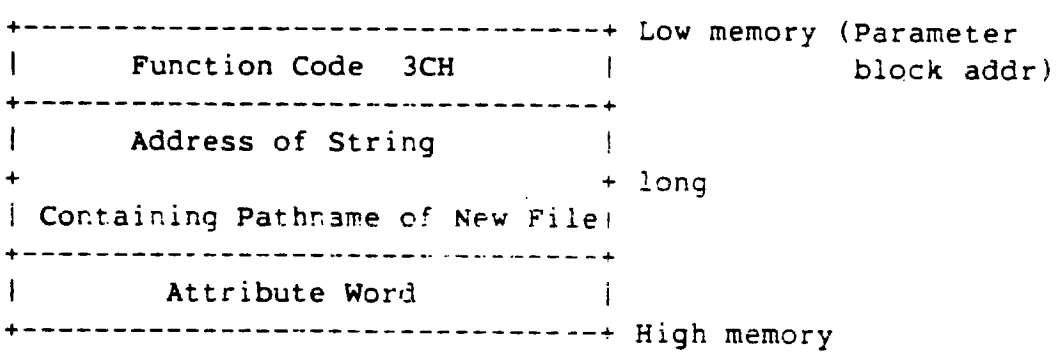
Return Parameters

Register D0.L : Contains 0 if operation succeeds,
non-zero if error occurs.

9.33. 3C: Create a File

```
    LONG
f_create (name,attr)
    BYTE        *name;
    WORD         attr;
{
}
```

Parameter Block



The parameter block contains a long and word.

Long is a pointer to a null-terminated string specifying the complete pathname of the new file.

Word indicates file attributes, depending on these values:

- 01H File set to read-only
- 02H File hidden from directory search
- 04H File set to system, hidden from directory search
- 08H File contains volume label in first 11 bytes

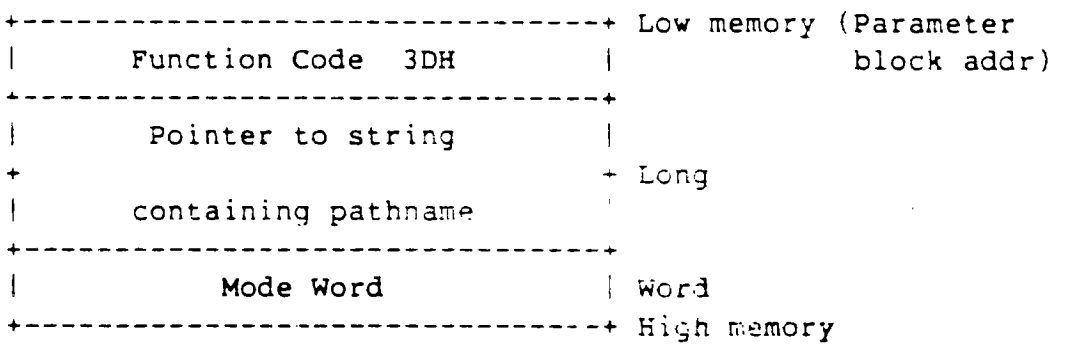
Return Parameters:

Register D0.L : Contains file handle if the file was created successfully, negative if an error occurred.

9.34. 3D: Open File

```
    LONG
f_open (pname,mode)
    BYTE    *pname;
    WORD mode;
{
}
```

Parameter Block



Long is a pointer to a null-terminated string containing the pathname of the file to open.

Word contains a code indicating the file read-write mode:

0 = file open for reading only
1 = file open for writing only
2 = file open for reading or writing

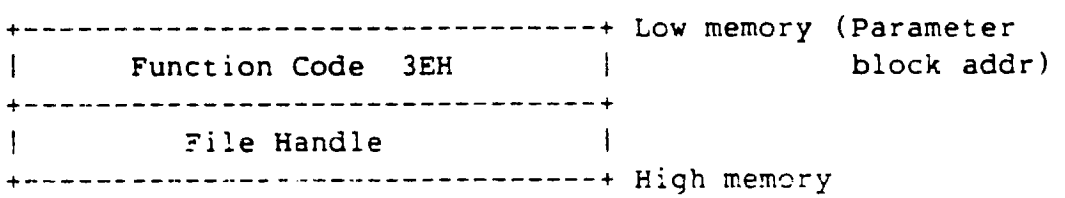
Return Parameters:

Register D0.L : Contains file handle if the file was opened successfully, negative if an error occurred.

9.35. 3E: Close File

```
    LONG
f_close (handle)
    WORD      handle;
{
}
```

Parameter Block



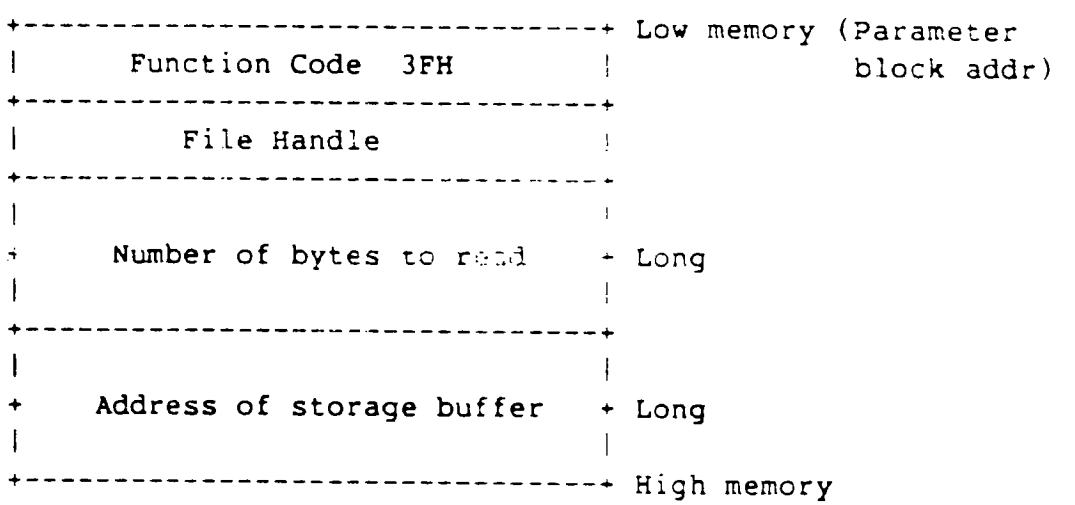
Return Parameters:

Register D0.L : Contains 0 if the file was closed successfully,
non-zero if an error occurred.

9.36. 3F: Read File

```
    LONG
f_read (handle, cnt, pbuffer)
    WORD       handle;
    LONG       cnt;
    BYTE       *pbuffer;
{
}
```

Parameter Block



Word contains the file handle.

Long 1 contains the number of bytes to read

Long 2 contains the buffer location to store the read bytes.

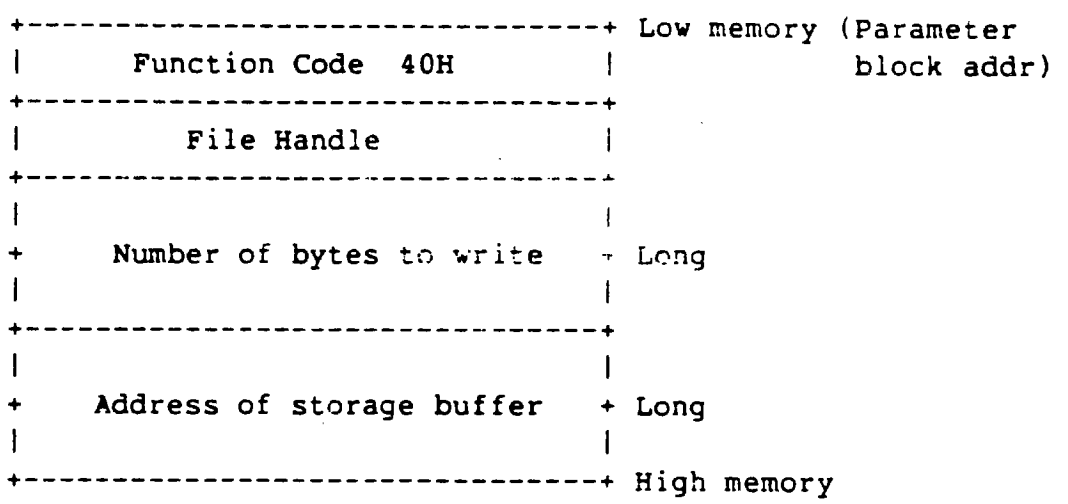
Return Parameters:

Register D0.L : Contains number of bytes read if read operation completed successfully, or an error code if an error occurred.

9.37. 40: Write File

```
    LONG
f_write (handle, cnt, pbuffer)
    WORD      handle;
    LONG      cnt;
    BYTE      *pbuffer;
{
}
```

Parameter Block



Word contains the file handle.

Long 1 contains the number of bytes to write
Long 2 contains the buffer location containing the bytes to write.

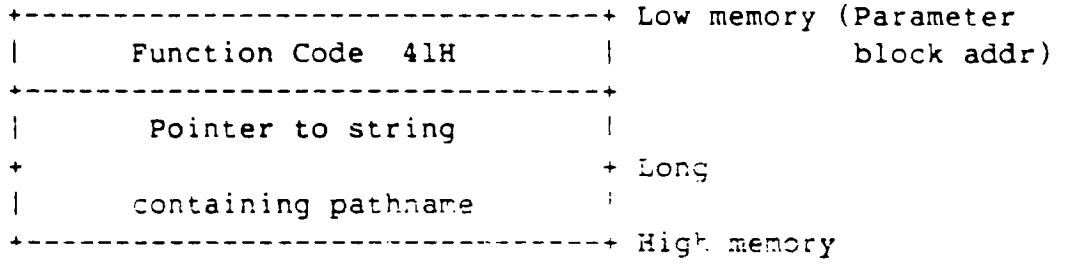
Return Parameters:

Register D0.L : Contains number of bytes written if write operation completed successfully, or an error code if an error occurred.

9.38. 41: Delete File

```
    LONG
f_delete (name)
    BYTE        *name;
{
}
```

Parameter Block



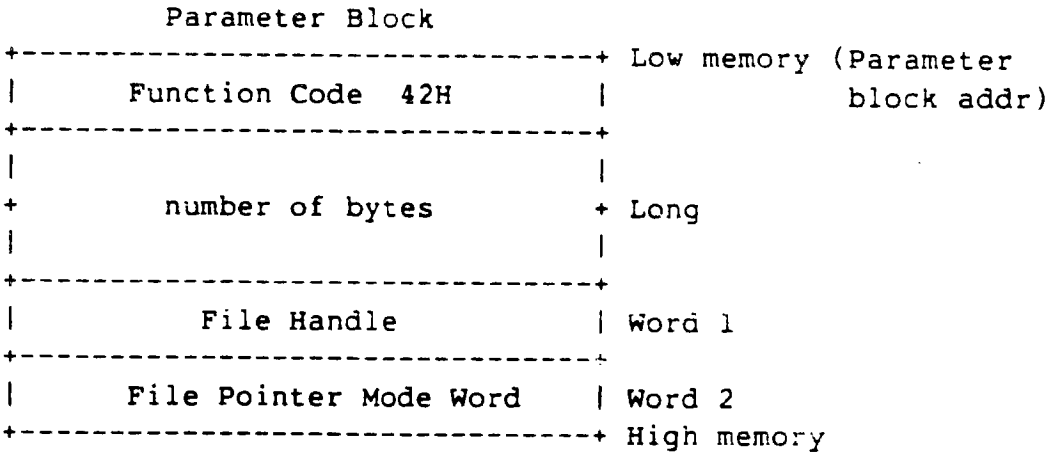
Long contains the address of a null-terminated string specifying the directory entry.

Return Parameters:

Register D0.L : Contains 0 if the file was removed, non-zero if an error occurred.

9.39. 42: Seek File Pointer

```
    LONG
f_seek (softs, handle, smode)
    LONG    softs;
    WORD    handle;
    WORD    smode;
{
}
```



Long contains N, the number-of-bytes argument. Long is signed; negative values are useful in Modes 1 and 2 below. Positive N moves toward end of file; negative N toward beginning of file.
Word 1 is the file handle.
Word 2 is the method used to move the file pointer:

Mode
- 0 move pointer to N bytes from beginning of file
- 1 move pointer N bytes from current location
- 2 move pointer to N bytes from end of file

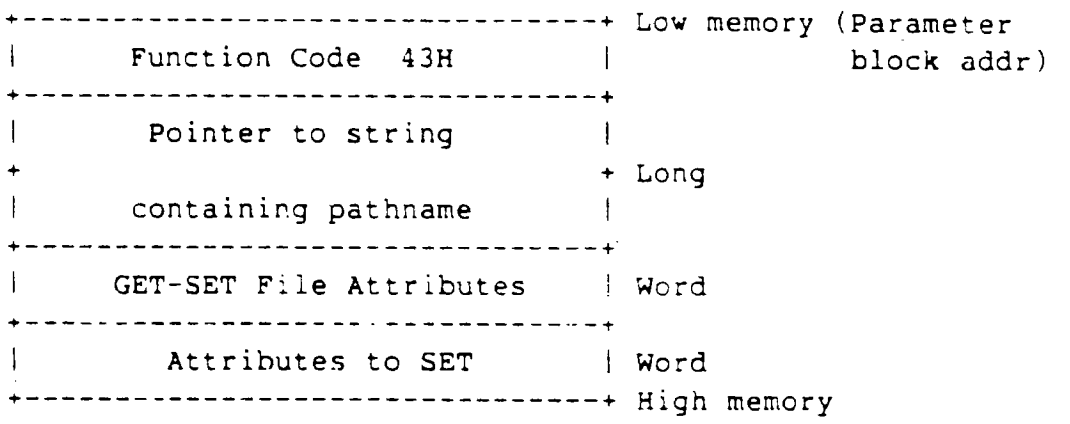
Return Parameters:

Register D0.L : Contains absolute file pointer location, as the number of bytes from the beginning of the file.

9.40. 43: Get/Set File Attributes

```
    LONG
f_attrib (p, wrt, mod)
    BYTE    *p;
    WORD    wrt,mod;
{
}
```

Parameter Block



Long contains the address of a null-terminated string containing the complete pathname of the specified file.

Word contains a 0 to get the file's attributes or a 1 to set the file's attributes Word indicates file attributes, depending on these values:

01H File set to read-only
02H File hidden from directory search
04H File set to system, hidden from directory search
08H File contains volume label in first 11 bytes
01H File is a subdirectory
01H File has been written to and closed

Return Parameters:

Register D0.L : If it is a Get Attributes operation, the current attributes are returned in D0.L.

9.41. 45: Duplicate File Handle

```
    LONG
f_dup (stdhnd)
    WORD stdhnd;
{
}
```

Input is a standard handle. Returns a non-standard handle that refers to the same file Error returns are EIHNDL and ENHNDL.

9.42. 46: Force File Handle

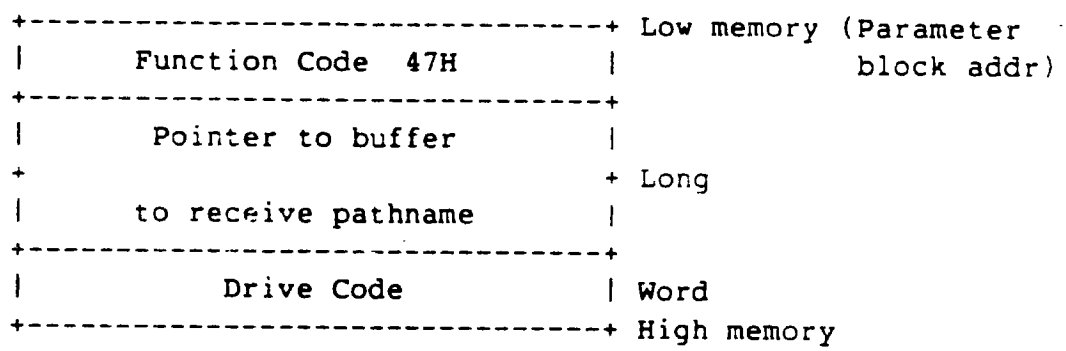
```
      LONG
f_force (stdhnd, nsthnd)
    WORD stdhnd, nsthnd;
{
}
```

Forces the standard handle to point to the same file or device as the non-standard handle. Returns E_OK on success, EIHNDL on failure.

9.43. 47: Get Current Directory

```
    LONG
d_getpath (pathbuf, drive)
    BYTE    *pathbuf;
    WORD    drive;
{
}
```

Parameter Block



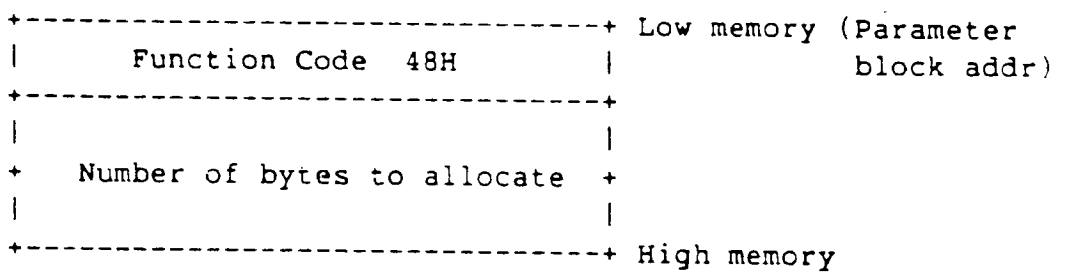
Long contains the address of a 64-byte buffer that receives the complete pathname of the current directory.

Word specifies the drive; 0 = default drive, 1 = A, 2 = B, etc.

9.44. 48: Allocate Memory

```
    LONG
m_alloc (nbytes)
    LONG    nbytes;
{
}
```

Parameter Block



Long contains the number of bytes to allocate, or -1L (FFFFFFFF), which returns maximum available memory.

Return Parameters:

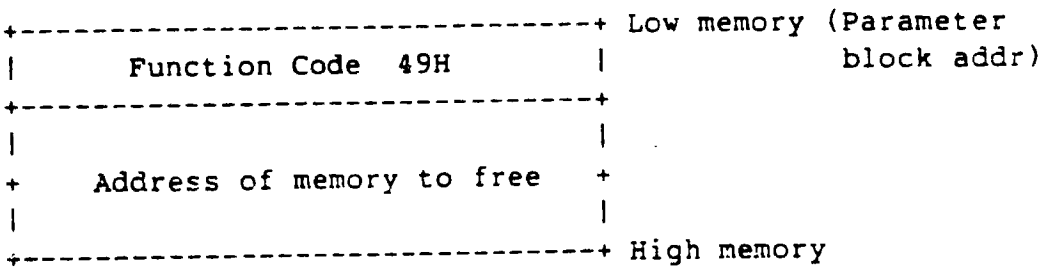
Register D0.L : If nbytes specifies the number of bytes to allocate, D0.L contains pointer to the starting address of the block of memory allocated. If allocation fails, D0.L contains 0.

If nbytes equals FFFFFFFF, D0.L returns the number of available bytes.

9.45. 49: Free Allocated Memory

```
    LONG
m_free (maddr)
    LONG    maddr;
{
}
```

Parameter Block



Long contains address of memory to free.

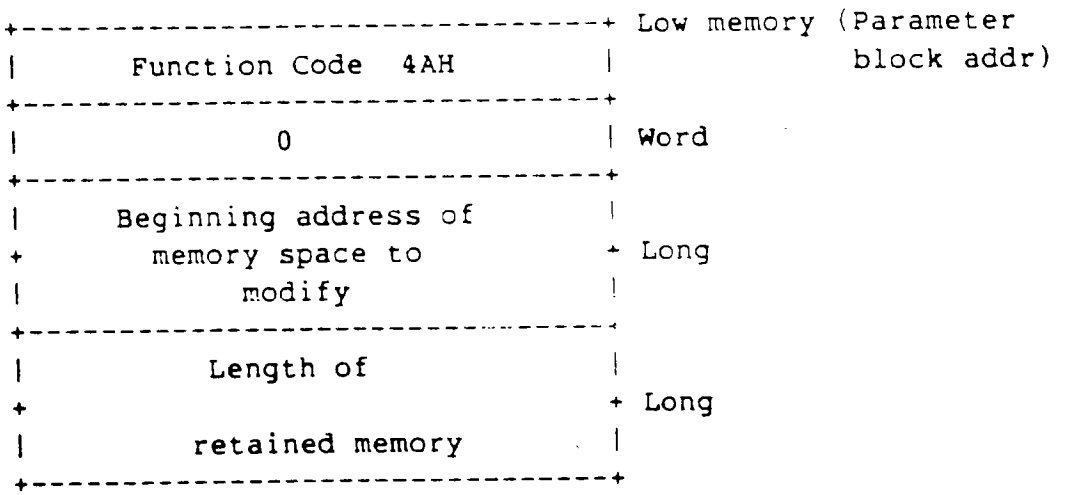
Return Parameters:

Register D0.L : Contains 0 if memory was freed, non-zero if an error occurred.

9.46. 4A: Shrink Size of Allocated Memory

```
    VOID
m_shrink (mp, size)
    BYTE *mp;
    LONG size;
{
}
```

Parameter Block



Function:

In the GEM DOS memory model, stack space grows downwards from high memory and program and data space grows upwards from low memory. The m_shrink function call polices memory space and reallocates unused memory for GEM DOS's use. Long 1 contains the beginning address of the memory to be returned to GEM DOS. Long 2 contains the length of the returned space. Word is reserved and must be zero.

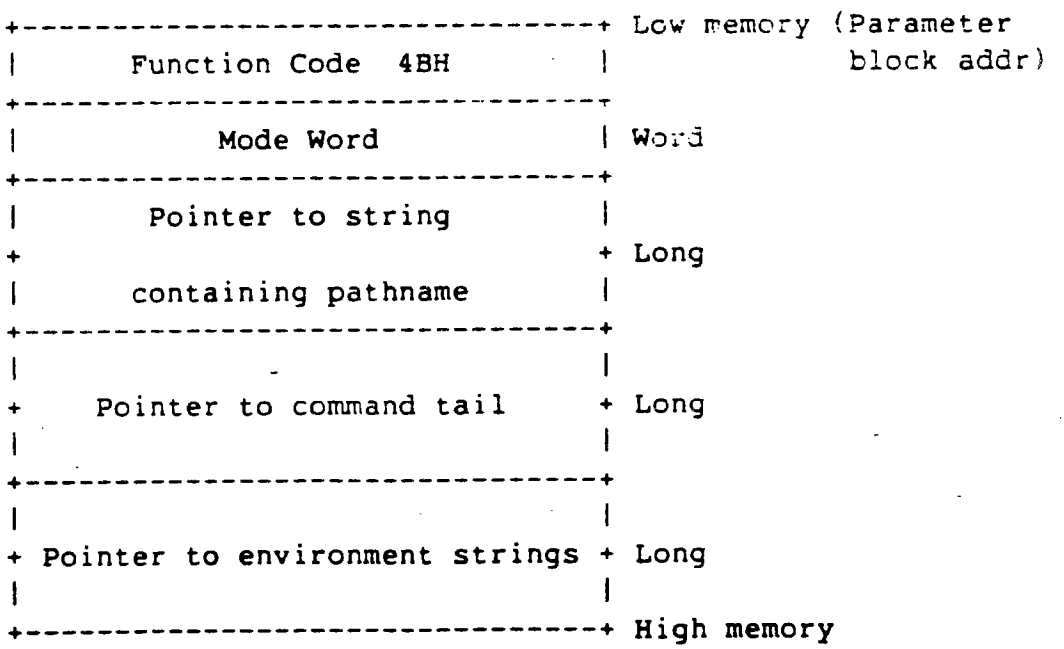
Return Parameters:

Register D0.L : Contains 0 if the block was adjusted successfully, non-zero if an error occurred.

9.47. 4B: Load or Execute a Process

```
    LONG
p_exec (load, pcspec, pcmdln, penvstr)
    WORD      load;
    BYTE      *pcspec;
    BYTE      *pcdln;
    BYTE      *penvstr;
{
}
```

Parameter Block



Word contains either a 0 or a 3, indicating these actions:

0 = load and execute the program
3 = load program but do not execute; used with overlays

Long 1 is a pointer to a null-terminated string specifying the name of the file to load.

Long 2 is a pointer to a command tail, which includes redirection details.

Return Parameters:

Register D0.L: For load only, D0.L returns address of base page. Contains base page address.

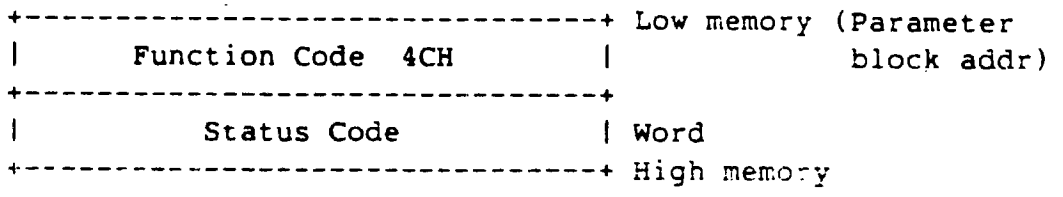
For load and execute, D0.L returns return code of child process upon child's termination.

If load fails, D0.L returns ERROR.

9.48. 4C: Terminate Process

```
    VOID
p_term (code)
WORD code;
{
}
```

Parameter Block



Word contains a status code that can be interrogated by the parent process.

Register D0.L : Contains 0 if the file was terminated successfully, non-zero if an error occurred.

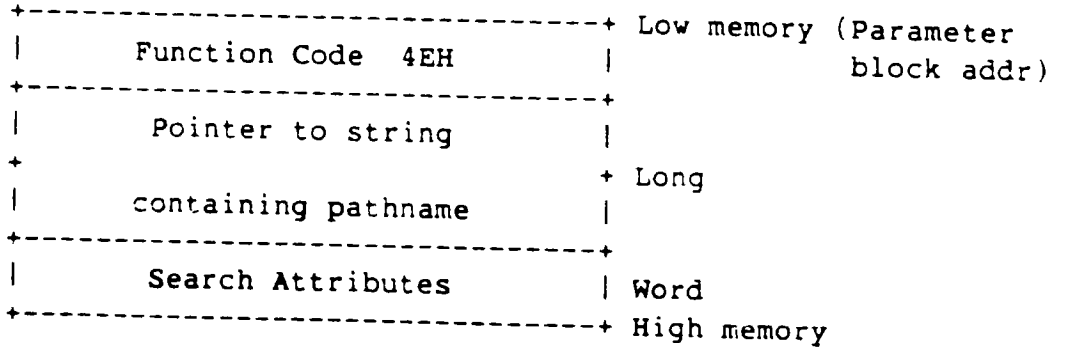
Function:

This system call terminates the current process and transfers control to the invoking process.

9.49. 4E: Search for First Occurance of Filespec

```
    LONG
f_sfirst (pspec, attr)
    BYTE      *pspec;
    WORD       attr;
{
}
```

Parameter Block



Long is a pointer to the null-terminated string specifying the file to find. May contain * or ? wildcards in filename, but not in path prefix.

Word contains a code specifying the search attributes, as shown below:

00H File is normal file entry
01H File is read-only
02H File hidden from directory search
04H File is system file
08H File is a volume label
10H File is a subdirectory
20H File has been written to and closed

The search procedure take these codes into account in this way:

If the attribute code is 00H find normal file entries only. No volume labels, subdirectories, hidden, or system files are accepted as matches.

If the attribute field is set for hidden or system files, they are included in the search set. To look at all directory entries except the volume labels, set the attribute bits for hidden, system, and directory all on.

If the attribute field is set for the volume label, the search only considers volume labels.

Return Parameters:

Register D0.L = E_OK if file found
= EFILNF if file not found

Function:

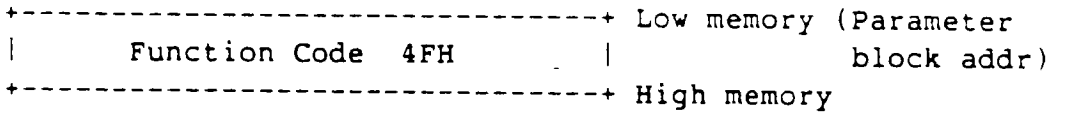
Searches for a match for the specified filename, according to the attribute bit settings described above. If a match is found, a 44-byte DMA buffer is formatted as follows:

Longs	Contents
0 - 20	Reserved for OS use
21	File attributes
22 - 23	File time stamp
24 - 25	File date stamp
26 - 29	Longword of file size
30 - 43	Name and extension of found file

9.50. 4F: Search for Next Occurrence of Filespec

```
    LONG
f_snext()
{
}
```

Parameter Block



R Parameters:

Register D0.L = E_OK if file found
= ENMFIL if no file found

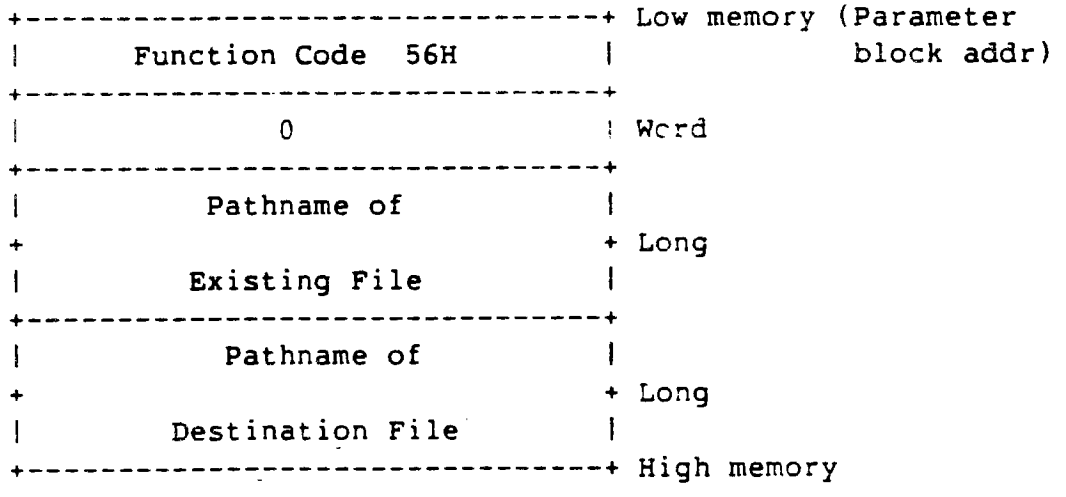
Function.

This system call uses the information specified in a previous Find Matching File system call to locate the next matching file. The DMA buffer bytes 0-20 must remain untouched from the previous SFIRST or SNEXT. If a file is found, the DMA buffer initialized in the Find Matching File system call is updated with the new filename.

9.51. 56: Rename a File

```
    LONG
f_rename (res, p1, p2)
    WORD   res;
    BYTE   *p1;
    BYTE   *p2;
{
}
```

Parameter Block



Word is reserved and must be zero.

Long 1 is a pointer to the pathname of the existing file

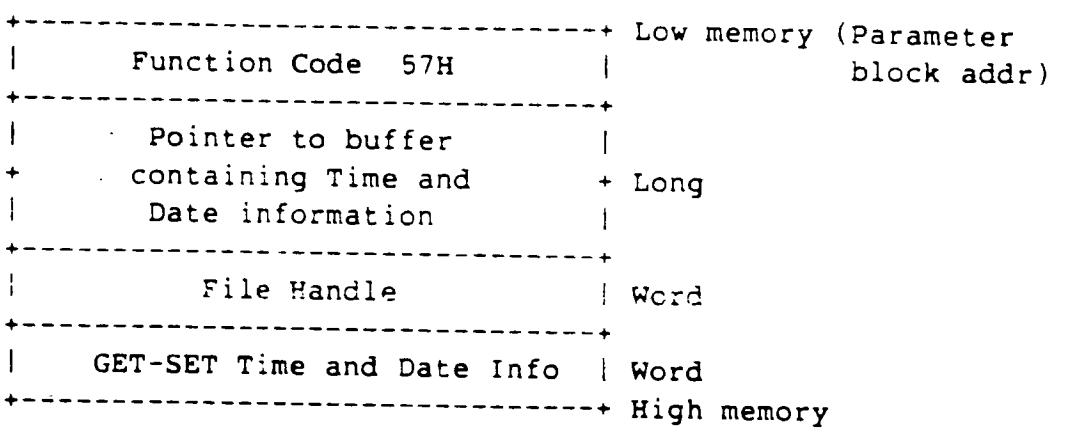
Long 2 is a pointer to the pathname of the destination file

The destination file must not exist; this instruction can move the file to another subdirectory on the same drive.

9.52. 57: Get/Set File Date & Time Stamp

```
    VOID
f_datime (h, buff, set)
    WORD     h;
    BYTE     *buff;
    WORD     set;
{
}
```

Parameter Block



Long is a pointer to a buffer that contains the data and time information.

Word 1 is the specified file handle.
Word 2 is a flag specifying setting or return of data and time information. If 0 set data and time; if 1, get data and time information. In either case, the buffer holds two words, with time first. The format of date and time is shown below:

Bits 0 through 4 indicate the date in the range 1 - 31
Bits 5 through 8 indicate the month in the range 1 - 12
Bits 9 through 15 indicate the year (since 1980) in the range 0-119

Bits 0 through 4 indicate the binary number of two-second increments
Bits 5 through 10 indicate the binary number of minutes
Bits 11 through 15 indicate the binary number of hours

10. BIOS Function Calls

10.1. Parameter Format In Memory

To preserve system security and integrity, the BIOS is callable only from supervisor mode. Applications that use the BIOS should follow these conventions, shown below in C:

```
return_value = trap13(function_number, parameter_1,
   parameter_2, ...);
```

where 'trap13' is defined as:

```
     trap13:
          move.l     (sp)+,retsave
          trap      #13
          move.l      retsave,-(sp)
          rts
```

The parameters will then be on the stack in the form:

```
          ...
          parameter_2
          parameter_1
     WORD      function_number
     LONG      PC
     WORD      status register
```

10.2. BIOS Initialization Sequence

Initialize exception vectors.
Disable interrupts.
Perform osinit().
Enable interrupts.

10.3. 00

```
VOID get_mpb(p_mpb)
MPB *p_mpb;
{
}
```

Upon entry, p_mpb points to a 'sizeof(MPB)' byte block to be filled in with the system initial Memory Parameter Block. Upon return, the MPB is filled in.

Formats are as follows:

```
MPB /* memory partition block */
{
        MD      *mp_mfl;    /* Point to MD described below */
        MD      *mp_mal;    /* 0L */
        MD      *mp_rover;  /* Point to same MD as mp_mfl */
} ;

MD /* memory descriptor */  /* Only 1 allowed in first release */
{
        MD      *m_link;    /* 0L */
        long    m_start;    /* Beginning address of free memory *
        long    m_length;   /* Number of bytes of free memory */
        PD      *m_own;     /* 0L */
} ;
```

10.4. 01

```
LONG character_input_status(h)
WORD h;
{
}
```

h is a character device handle that specifies one of the following devices:

0 PRN:
1 AUX:
2 CON:

Returns status in D0.L:

-1 device is ready
0 device is not ready

10.5. 02

```
 LONG character_input(h)
WORD h;
{
}
```

h is a character device handle described in Function 01.

This function does not return until a character has been input.
It returns the character value in D0.L, with the high word set to zero.

For CON:, it returns the GSX 2.0 compatible scan code in the low byte of the high word, and the ASCII character in the lower byte, or zero in the lower byte if the character is non-ASCII

For AUX:, it returns the character in the low byte.

10.6. 03

```
 VOID character_output(h, char)
WORD h, char;
{
}
```

h is a character device handle described in Function 01.
char is a character in the low 8 bits of the word.

This function does not return until the character has been output.

10.8. 05

```
 LONG set_exception_vector(vecnum, vecadr)
WORD vecnum;
LONG vecadr;
{
}
```

vecnum is the number of the exception vector to get or set
vecadr is the long address to set into the exception vector table.
No set is done if vecadr is -1.

Vectors 0x00 - 0xFF are defined by the 68000 hardware. GEM DOS has defined extended vectors as follows:

0x100	Timer Tick
0x101	Critical Error Handler
0x102	Terminate Handler
0x103 - 0x1FF	Reserved for future use by GEM DOS
0x200 - 0x2FF	Reserved for user defined vectors

Function returns a long address that was the previous entry.

Notes on handling extended vectors:

0x100 Timer Tick

It is the responsibility of the BIOS to save all registers before it starts down this chain. There is 1 parameter on the stack; a WORD value of the number of milliseconds since the last tick. It is the responsibility of the handler that installs itself to jump to the previous handler in this vector.

0x101 Critical Error Handler

It is the responsibility of the handler that installs itself to save d3-d7/a3-a6 if they are used. The error number is a WORD parameter on the stack.

To ignore an error, set D0.L to 0.
To retry, set D0.L to 0x10000.
To abort, move parameter to D0 and sign extend it to a longword.

10.7. 04

```
LONG read_write_sectors(wrtflg, buffer, num, recn, drive)
WORD wrtflg;
char *buffer;
WORD num;
WORD recn;
WORD drive;
{
}
```

```
wrtflg   is $0 for read, $1 for write
buffer   is a long pointer to a byte address for the disk tr
   num   is the number of sectors to transfer
  recn   is the beginning record number to transfer
 drive   is 0 for drive A, 1 for drive B. ...
```

Function returns a 2's complement error number in D0.L.

It is the responsibility of the driver to check for media change before any write to FAT sectors. If media has changed, no write should take place, just return with error code.

0x102 Terminate Handler

It is the responsibility of the handler that installs itself to determine whether it should allow the termination to continue. To terminate, simply RTS. Otherwise, "longjump" back to the top of the application that installed the handler.

10.9. 06

```
LONG get_timer_ticks()
{
}
```

Returns the nearest number of milliseconds per tick in D0.L.

10.10. 07

```
BPB *get_bpb(d)
WORD d;
{
}
```

d is 0 for drive A, 1 for drive B, ...

Returns a pointer to the BIOS Parameter Block for the specified drive in D0.L. If necessary, it should read boot header information from the media in the drive to determine BPB values.

10.11. 08

```
LONG character_output_status(h)
WORD h;
{
}
```

h is a character device handle described in Function 01.

Returns status in D0.L:

-1	device is ready
0	device is not ready

10.12. 09

```
LONG media_change(d)
WORD d;
{
}
```

d is 0 for drive A, 1 for drive B, ...

Returns media change status for specified drive in D0.L:

0	Media definitely has not changed
1	Media may have changed
2	Media definitely has changed

10.13. 0A

```
LONG get_drive_map()
{
}
```

It returns a longword containing a bit map of logical drives in the system with bit 0, the least significant bit, corresponding to drive A.

Note: If the BIOS supports logical drives A and B on a single physical drive, it should return both bits set if a floppy disk is present.

11. The GEM-GSX Graphics Interface

The GEM-GSX graphics interface is documented in the manual set shipped with Digital Research's GEM product. Please refer to that documentation set for further information.

Multitasking

Two multitasking schemes are proposed. Both maintain complete process models in memory. Both types can exist concurrently within the system.

The first type of multitasking is a background activity (such as communications or printer spooling) that is interrupt driven and requires only a small percentage of the total processing time. Such tasks can run concurrently with a foreground process. Only one background process can run at any given time.

The second type of multitasking allows two or more processes in the foreground at once, with all processes suspended but the one in the active window. The active window is defined by the cursor position - moving the cursor from one window to another freezes the process the cursor leaves and activates the one it enters. When activated, a process retrieves an intertask communication from a designated pipe or queue. The number of applications running in this mode are limited only by available memory.

Real Time Response

The design goal for handling real-time interrupts and exception requests is 100 microseconds or less. This response speed supports foreground communications applications running at XXX baud.