GEMDOS Technical Specifications Version 8

GEMDOS Spec Version 8

Revision History

Version 6 contains the complete object module format material, the hardware requirements, and the development environment.

Version 7 contains corrections from the 2-20 meeting.

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

Sections needing information or completion.

Tools section (to come)
MINCE note

4.0 Memory Models

5.0 Base Page Format

6.0 System Status Codes

7.6 AUX_IN C binding
7.7 AUX_OUT C binding
7.8 PRINT-OUT C binding
7.10 DIRECT CON_IN C binding
7.11 CON_IN (No echo) C binding
7.15 AUXIN_STAT function number
7.16 AUXOUT_STAT function number
7.17 CONOUT_STAT function number
7.18 PRTOUT_STAT function number
7.19 Reset Disk C binding
7.21 Current Disk C binding
7.22 Set Disk Transfer Address C binding
7.23 Get Date C binding
7.24 Set Date C binding
7.25 Get Time C binding
7.26 Set Time C binding
7.28 Get Version Number C binding
7.29 Keep Process C binding
7.30 Get Disk Free Space C binding
7.45 SETBLOCK C binding

8.8 Unresolved comment about parameters

8.13 Extended functions undefined
8.14 Timer tick section has outstanding issues.

1. Hardware Requirements (Minimum and Optimum)

The GEMDOS 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

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

- 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

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-M68K, 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.

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

3. Object File Format

3.1. Advantages of the GEMDOS Object File Format

Presently, in the CP-M68K environment, the relocation information takes up the same amount of space in the object file as the text and data. That is, there is one word of relocation information for each word of text and data. The new format should reduce the relocation information to about 1-20th the size of the text and data, on the average.

The new format can be used for linked programs, and it allows symbol table references.

The format is easy for loaders to read, as all relocation entries occupy integral numbers of bytes.

The format can be used in systems where text, data, and bss sections are allocated separately at load time.

The format supports future processors, such as the 68020, which allow addresses to fall on odd boundaries.

3.2. Changes to the File Header

The file header number is 601C rather than 601A. In all other respects, it is identical to the old file header.

3.3. Relocation Information

The new format is a subset of the old format, with a few features deleted due to lack of use. Specifically, the deleted features are:

1) 16-bit relocatable addresses

2) Relocation types that required no action: absolute, PC-relative, first word of instruction.

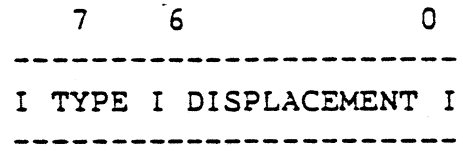
There is one relocation entry for each relocatable address in the text and data sections that needs a fixup. Each entry needs from one (most commonly) to seven bytes. The relocation entry contains:

1) A **type-displacement** byte (always present)

2) An optional **displacement extension**, consisting of 1,2, or 4 bytes.

3) An optional **symbol table index** (2 bytes).

Type-Displacement Byte



The **displacement** field is the offset added to the previous relocatable address to obtain the current, desired address. If this is the first such relocation entry in the file, adding the displacement value to the starting address of the file yields the address of the first relocatable address.

Because the contents of the displacement offset is limited to seven bits, some other scheme must be used to support displacements greater than 127 bytes. Accordingly, the displacement field has the following meaning:

0	End of relocation information; the relocation record is a dummy and does not perform a fixup.
1	The actual displacement is contained in an unsigned byte immediately following the type-displacement byte.
2	The actual displacement is contained in an unsigned word immediately following the type-displacement byte.
3	The actual displacement is contained in an unsigned 32-bit long word immediately following the type-displacement byte.
4 through 127	The displacement is contained in the type-displacement byte itself.

The extra bytes following the type-displacement byte when the displacement value is 1,2, or 3 are called the displacement extension bytes.

The **type** field has the following meaning:

0 Normal load-time relocation record.

1 Symbol reference. In this case, the relocatable address is an offset relative to a symbol in the object modules's symbol table. The index into the symbol table is a 16-bit word that follows the displacement extension bytes.

A symbol reference will never appear in a loadable, fully-linked program.

3.4. Notes For Writing Loaders

The object module itself contains the text and data sections in a contiguous block; all addresses in the program are virtual addresses pointing into a program image whose base address is the text starting address given in the header.

For example, if the starting virtual address of the text section is 0x400, and the length of the text section is 0x800, then the data section starts at virtual address 0xC00. If the address 0xD00 subsequently appears in the program, the address refers to a location offset 0x100 from the start of the data section.

If the text, data, and BSS are loaded in non-contiguous blocks, extra care must be taken when the relocation information crosses the boundary between the text and data sections.

There is no explicit information in the relocation information that indicates when a boundary is crossed. The loader must check its pointer to the text and data sections during relocation each time it adds the displacement field, and it must adjust the pointer if it has crossed over from text to data.

4. CCP Commands

Built-In Commands

LS
DIR
CAT
CD
MD
RD
RM
DEL
REN
SHOW
INIT
COPY
GET TEMPORARY
PUT TEMPORARY
BREAK TEMPORARY: For debugging
EXIT Terminate CLI and return to parent process

6. Base Page Format

```
PD      /* BASEPAGE FORMAT */

{
/* 0x00 */
long     p_lowtpa;      /* beginning of TPA */
long     p_hitpa;       /* end of TPA + 1 */
long     p_tbase;       /* text (code) base */
long     p_tlen;        /* text (code) length */
/* 0x10 */
long     p_dbase;       /* data base */
long     p_dlen;        /* data length */
long     p_bbase;       /* bss base */
long     p_blen;        /* bss length */
/* 0x20 */
char     *p_xdta;       /* disk transfer address */
PD       *p_parent;     /* parent PD */
char     p_lddrv;
char     p_curdrv;      /* current drive */
char     p_uft[NUMSTD];/* index into sys file table for std files
/* 0x30 */
char     p_curdir[16]; /* index into sys dir table"
```

7. System Status Codes

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

If the upper word in D0.L is FFFF, the content of the lower word is the returned code, but this code must be NEGATED to obtain the correct error code.

When D0.L holds a word or byte value returned as a status code, the upper word is set to 0000 and the lower word contains the returned information.

7.0.1. Error Codes

OK	0
Error	-1
Drive not ready	-2
Unknown command	-3
CRC error	-4
Bad request	-5
Seek error	-6
Unknown media	-7
Sector not found	-8
No paper	-9
Write fault	-10
Read fault	-11
General mishap	-12
Write protect	-13
Media change	-14
Unknown device	-15

8. System Function Calls

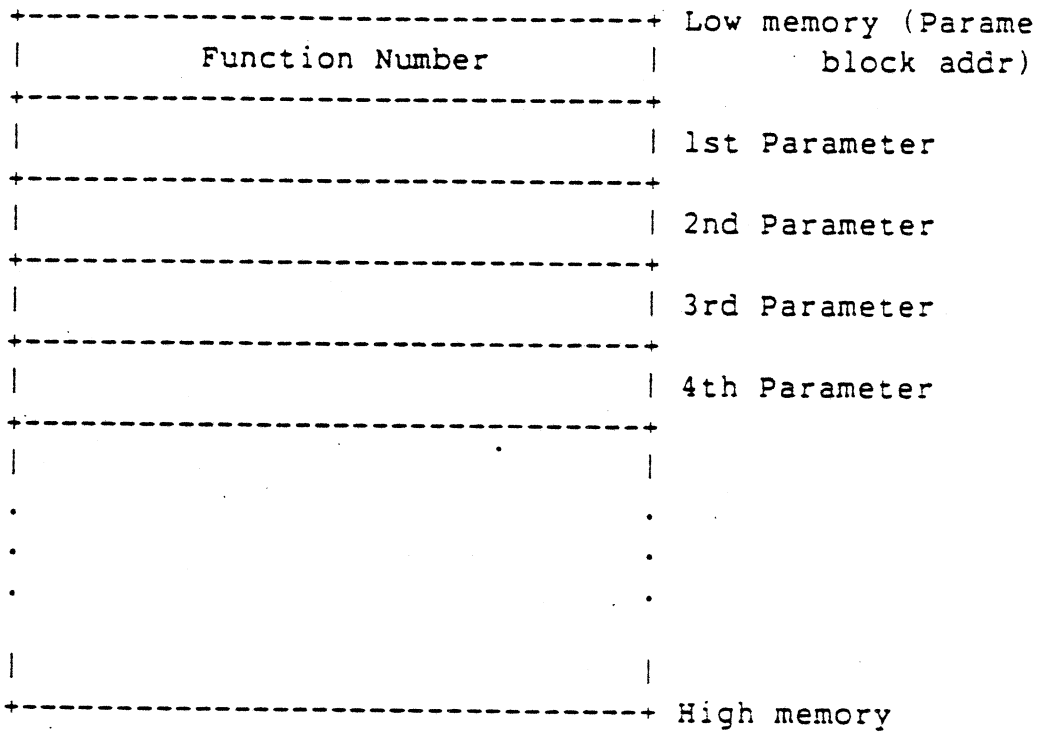
8.1. GEMDOS Parameter Passing Conventions

The GEMDOS 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 GEMDOS 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.

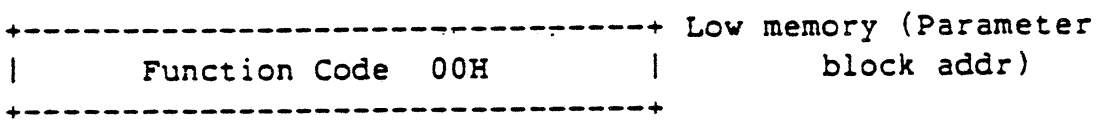
8.2. Returned Data

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.

8.3. 00: TERM

```
   VOID
xterm()
{
}
```

Parameter Block



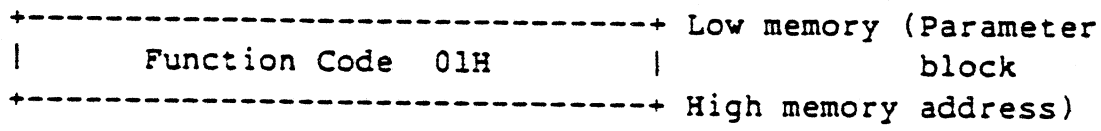
Function:

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

8.4. 01: CONIN

```
   LONG
conin()
{
}
```

Parameter Block



Return Parameters:

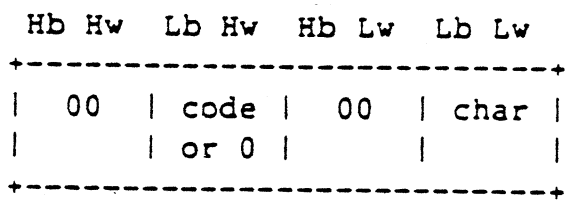
Register D0.L : Character read

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 emulates GSX 2.0, the console scan code is found in the low byte of the high word, as shown below. This byte is set to 00 otherwise.

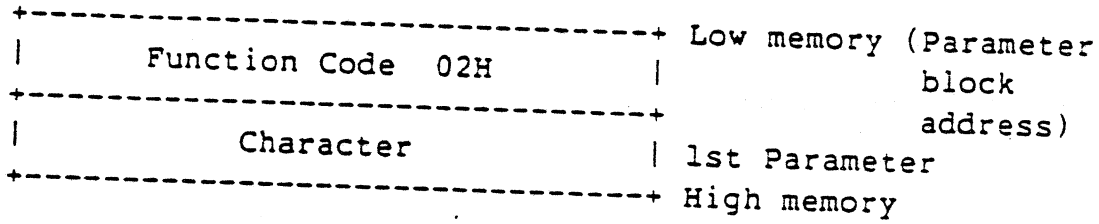
D0 Contents



8.5. 02: CONOUT

```
    VOID
conout(ch)
    WORD    ch;
{
}
```

Parameter Block



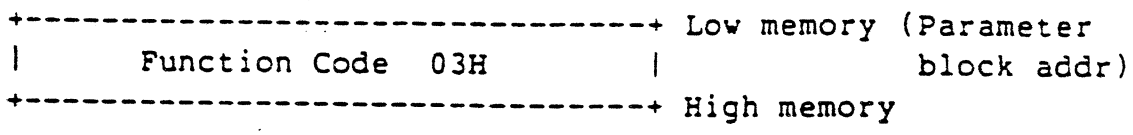
Function:

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.

8.6. 03: Auxiliary Input

Parameter Block



Return Parameters:

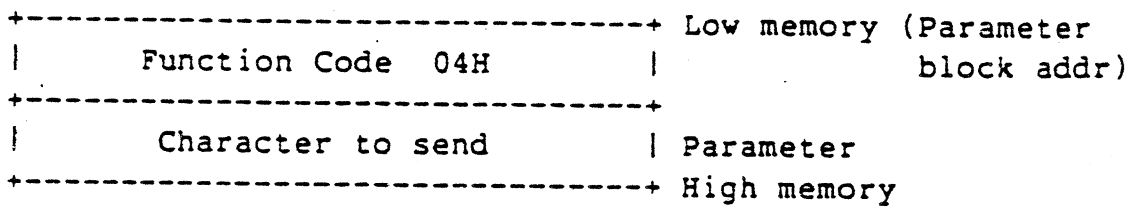
Register D0.L : Character read from auxiliary port.

Function:

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

8.7. 04: Auxiliary Output

Parameter Block

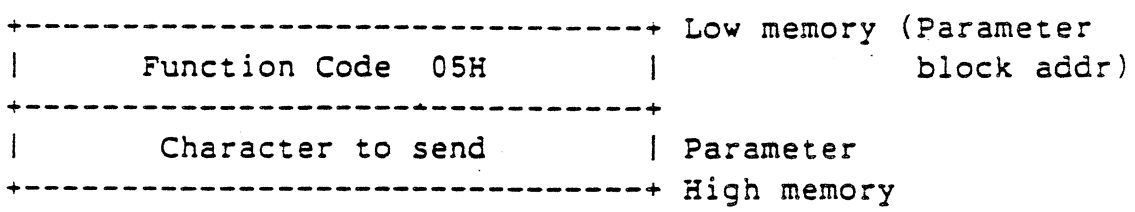


Function:

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.

8.8. 05: Printer Output

Parameter Block



Register D0.W : Character to send to printer.

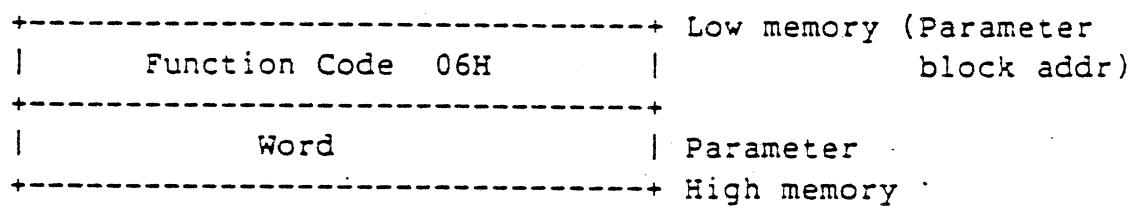
Function:

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.

8.9. 06: RAWCONIO

```
    LONG
rawconio(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.

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

Return Parameters:

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

8.10. 07: Direct Con In Without Echo

Parameter Block

Return Parameters:

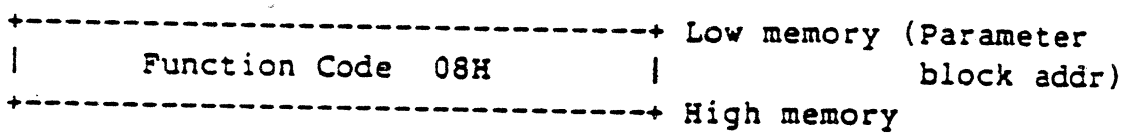
Register D0.L : Character read from standard input device.

Function:

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

8.11. 08: Con In Without Echo

Parameter Block



Return Parameters:

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

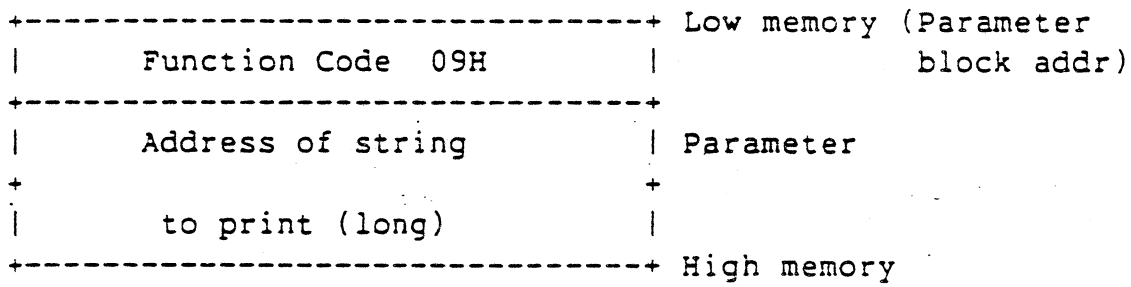
Function:

Read character from standard input device without echoing it to standard output device. Control characters are interpreted and have their proper effect.

8.12. 09: PRT_LINE

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

Parameter Block



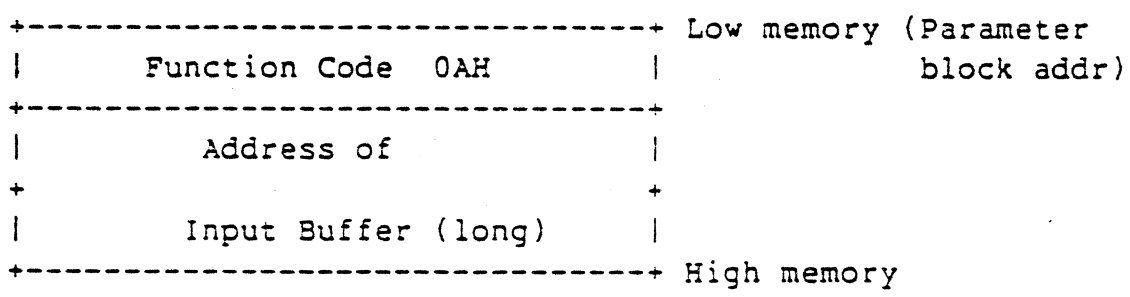
Function:

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

8.13. 0A: READLINE

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

Parameter Block



Function:

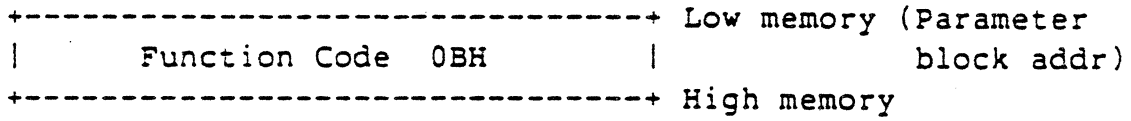
Sets address of keyboard input buffer and reads a line from the keyboard.

The first byte of the buffer contains the buffer size n, in bytes, and the second byte is set by GEMDOS, upon return, to the number of characters actually read. Character are read and stored from the third to n-1 bytes or until ENTER is read. If the buffer fills up to n-1, any character read but Enter causes the console bell to ring. The last character in the buffer is the carriage return (0DH) generated by ENTER.

8.14. 0B: CONSTAT

```
    LONG
constat()
{
}
```

Parameter Block



Return Parameters:

Register D0.W : Contains FFFF if a character is available;
0000 if no character available.

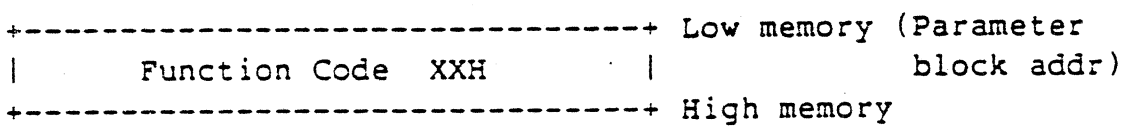
Function:

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

8.15. XX: AUXIN_STAT

```
    LONG
constat()
{
}
```

Parameter Block



Return Parameters:

Register D0.W : Contains FFFF if a character is available;
0000 if no character available.

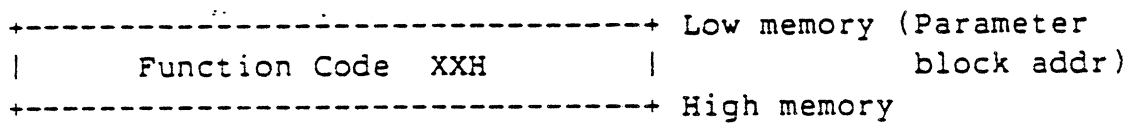
Function:

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

8.16. XX: AUXOUT_STAT

```
    LONG
AUXSTAT()
{
}
```

Parameter Block



Return Parameters:

Register D0.W : Contains FFFF if the device is ready to receive a character;
0000 if it is unavailable.

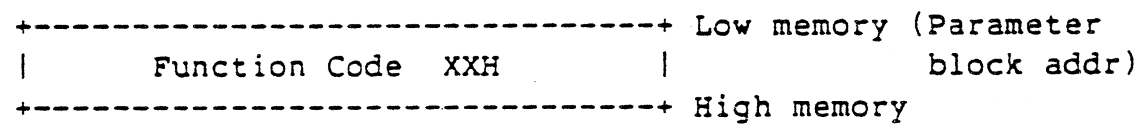
Function:

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

8.17. XX: CONOUT_STAT

```
    LONG
conout_stat()
{
}
```

Parameter Block



Return Parameters:

Register D0.W : Contains FFFF if the console is ready to receive a character;
0000 if it is unavailable.

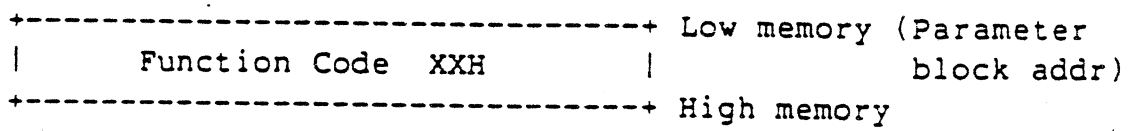
Fu on:

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

8.18. XX: PRTOUT_STAT

```
    LONG
prtout_stat()
{
}
```

Parameter Block



Return Parameters:

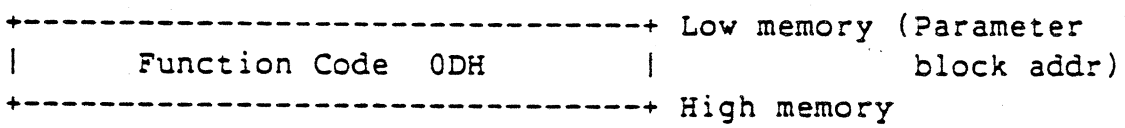
Register D0.W : Contains FFFF if the printer is ready to receive a character;
0000 if it is unavailable.

Function:

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

8.19. 0D: Reset Disk

Parameter Block



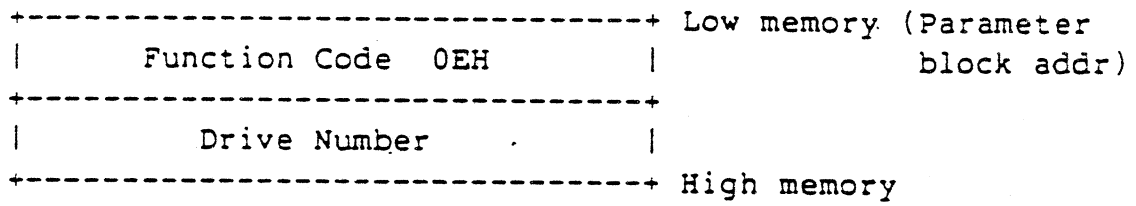
Function:

Disk reset. All file buffers are flushed, files are closed, and the directory is updated. When this drive is next accessed, the disk is logged in again.

8.20. 0E: SETDRV

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

Parameter Block



Drive Number:

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

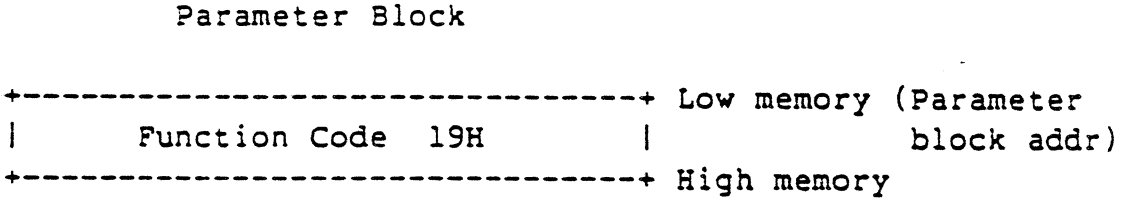
Return Parameters:

Register D0.L : Number of drives in system.

Function:

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

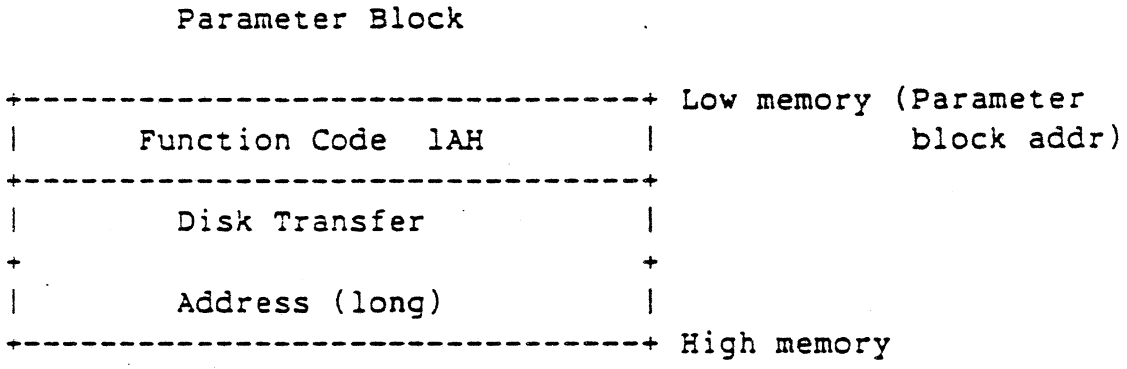
8.21. 19: Current Disk



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

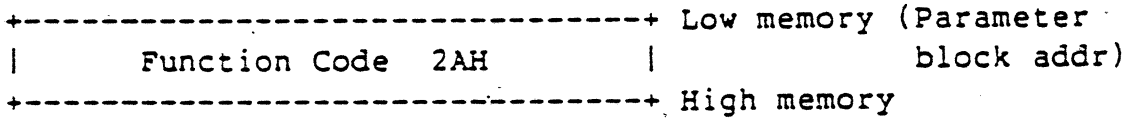
8.22. 1A: Set Disk Transfer Address

Register D1.L : Pointer to a long with with address of disk transfer address.



8.23. 2A: Get Date

Parameter Block



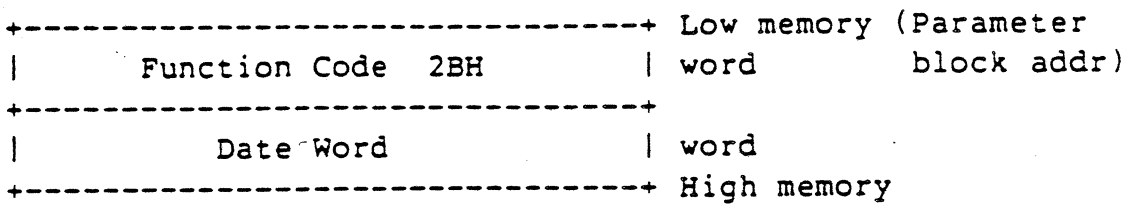
Ret Parameters:

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

8.24. 2B: Set Date

Parameter Block

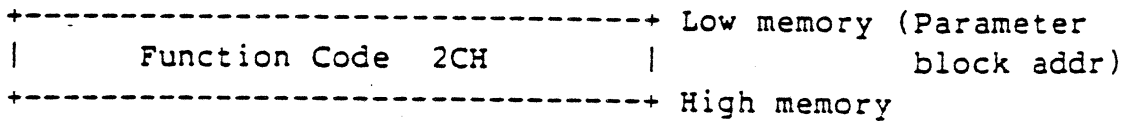


Date Word contains the new date in the format:

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

8.25. 2C: Get Time

Parameter Block



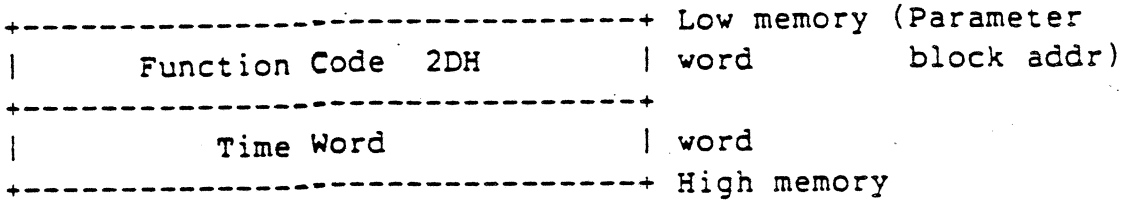
Return Parameters:

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

8.26. 2D: Set Time

Parameter Block



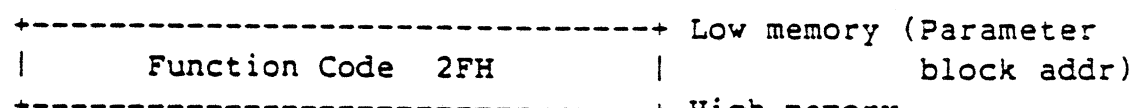
Time Word contains the new time in the format:

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

8.27. 2F: GETDTA

```
    LONG
getdta()
{
}
```

Parameter Block

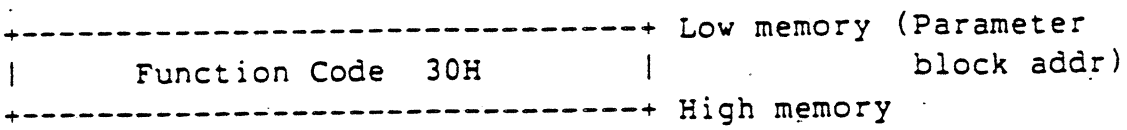


Return Parameters:

Register D0.L : Contains the current Disk Transfer Address.

8.28. 30: Get Version Number

Parameter Block



Return Parameters:

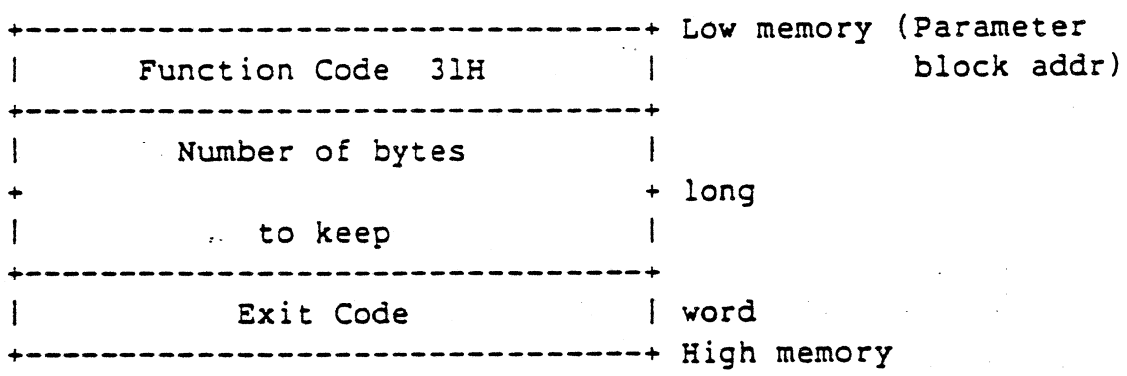
Register D0.W : Contains version number. For first release, this number will 0100H.

Function:

Lower byte contains minor version number; upper byte contains major version number. 0100 = version 1.00.

8.29. 31: Keep Process (Terminate and Stay Resident)

Parameter Block



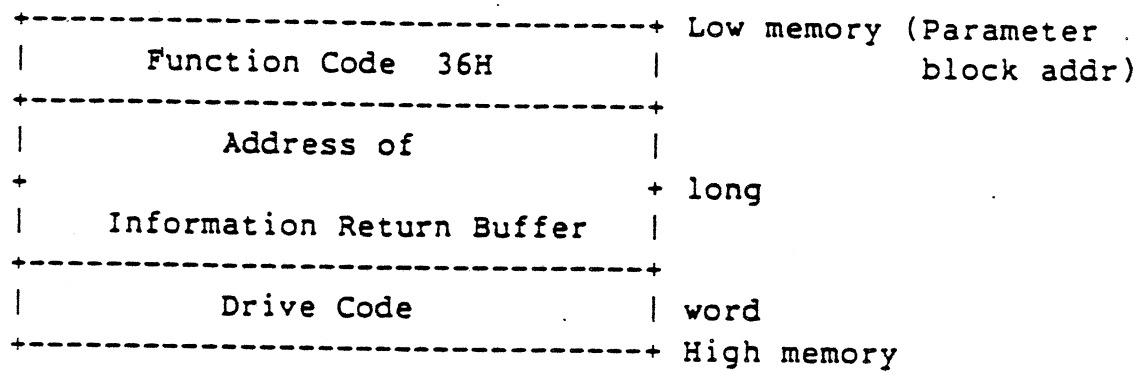
Long: Contains number of bytes to keep
Word: Contains exit code:

0 = success
1 = failure

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

8.30. 36: Get Disk Free Space

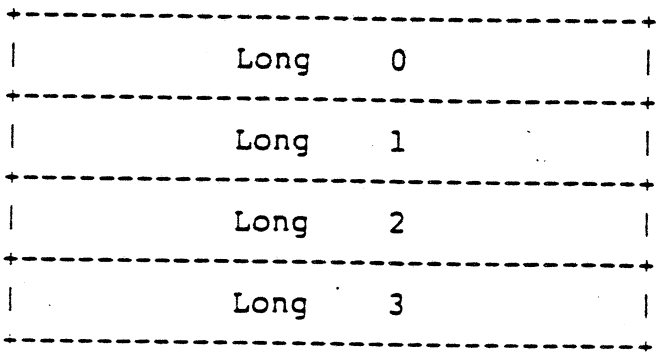
Parameter Block



Drive Code: 0 = the default drive, 1 = A, 2 = B, etc.

The Information Return Buffer holds the data retrived by this function call. The format of the IRB is:

Information Return Block



The Information Return Block contains 4 longs:

Long 0: Number of free allocation units on the specified drive.

Long 1: Total number of allocation units on the specified drive.

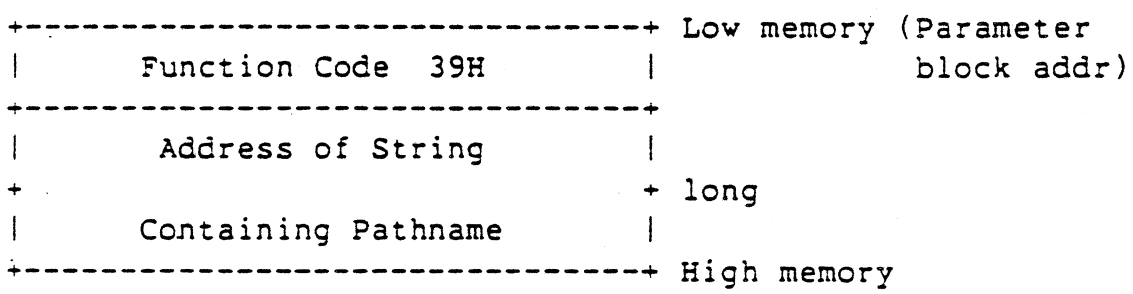
Long 2: Number of bytes per physical sector.

Long 3: Number of physical sectors per drive allocation unit.

8.31. 39: MKDIR

```
    LONG
mkdir(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.

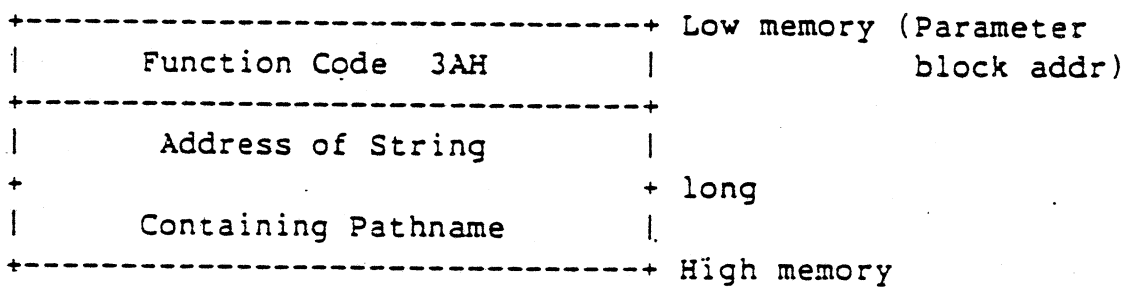
Return Parameters:

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

8.32. 3A: RMDIR

```
    LONG
rmdir(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.

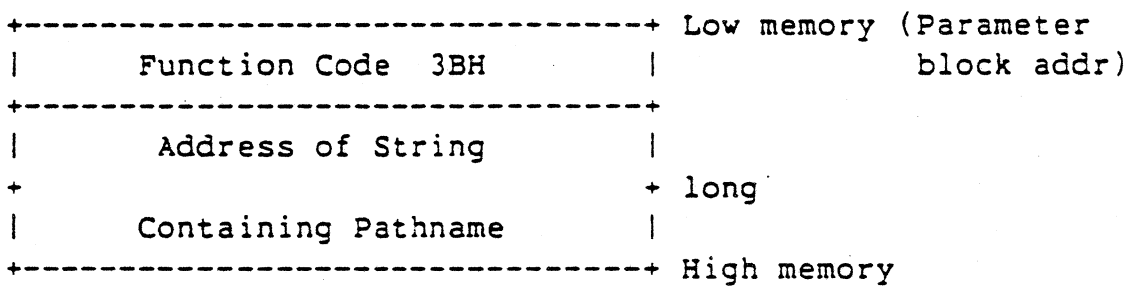
Function:

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

8.33. 3B: CHDIR

```
    LONG
chdir(pdrvpath)
    BYTE      *pdrvpath;
{
}
```

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.

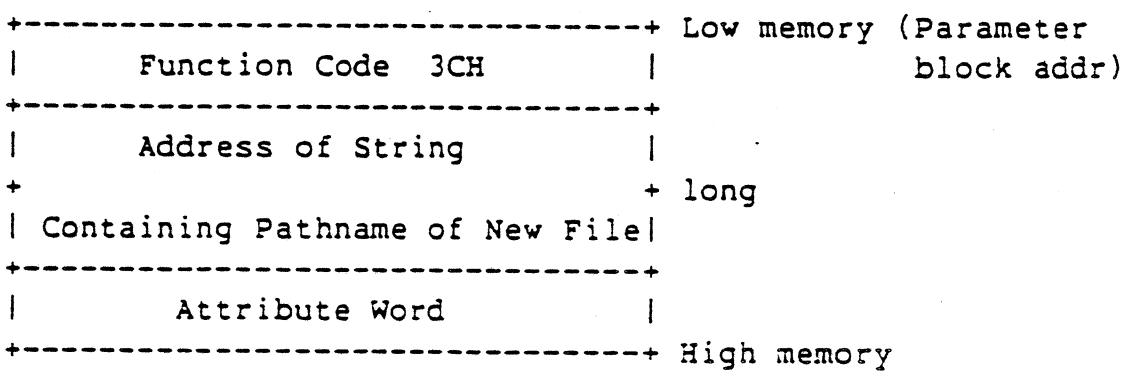
Return Parameters:

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

8.34. 3C: CREATE

```
    LONG
creat(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:

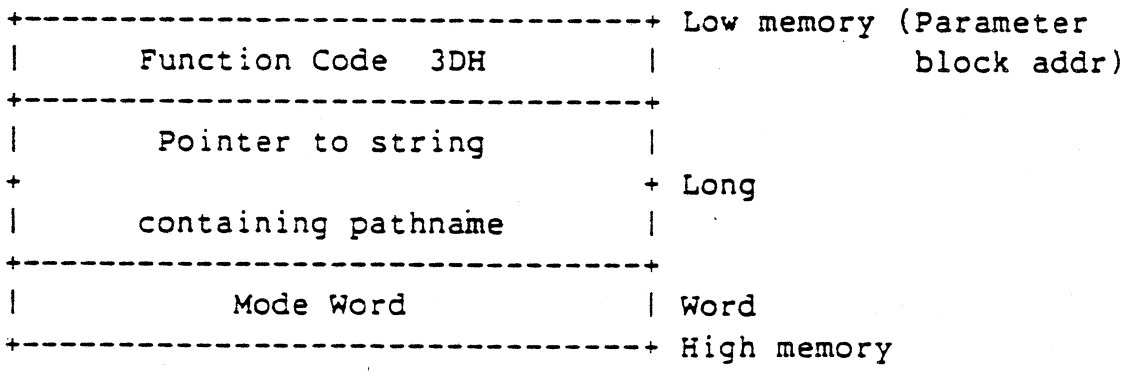
Register D0.L : Contains file handle if the file was created successfully,

negative if an error occurred.

8.35. 3D: OPEN

```
    LONG
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

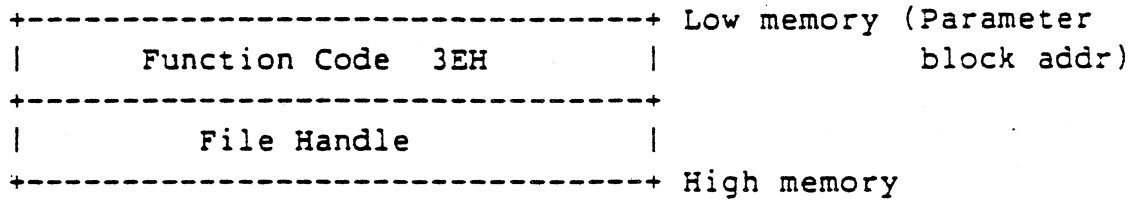
Return Parameters:

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

8.36. 3E: CLOSE

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

Parameter Block



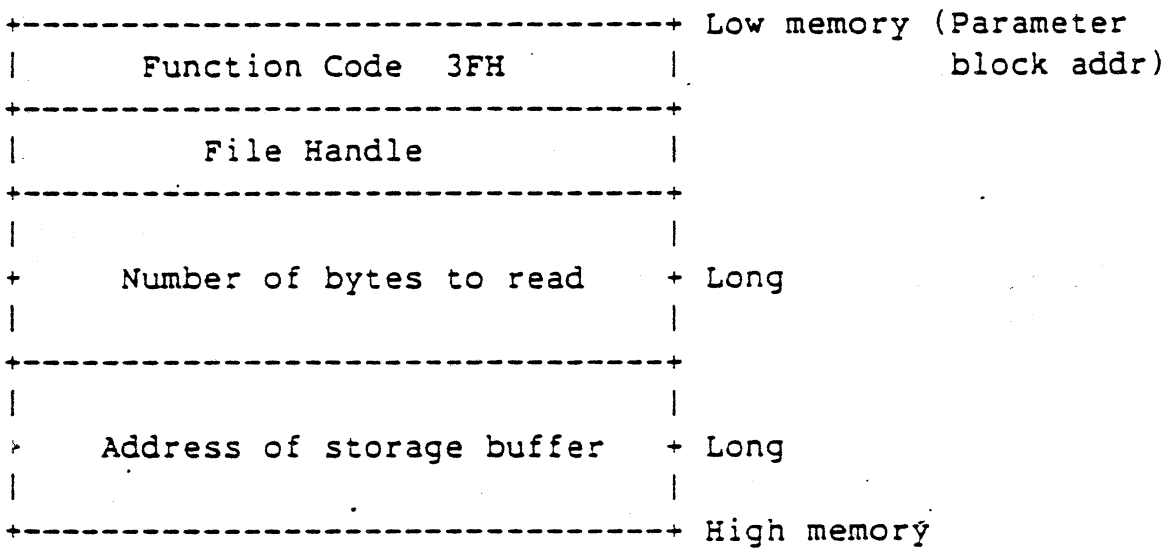
Return Parameters:

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

3.37. 3F: READ

```
   LONG
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:

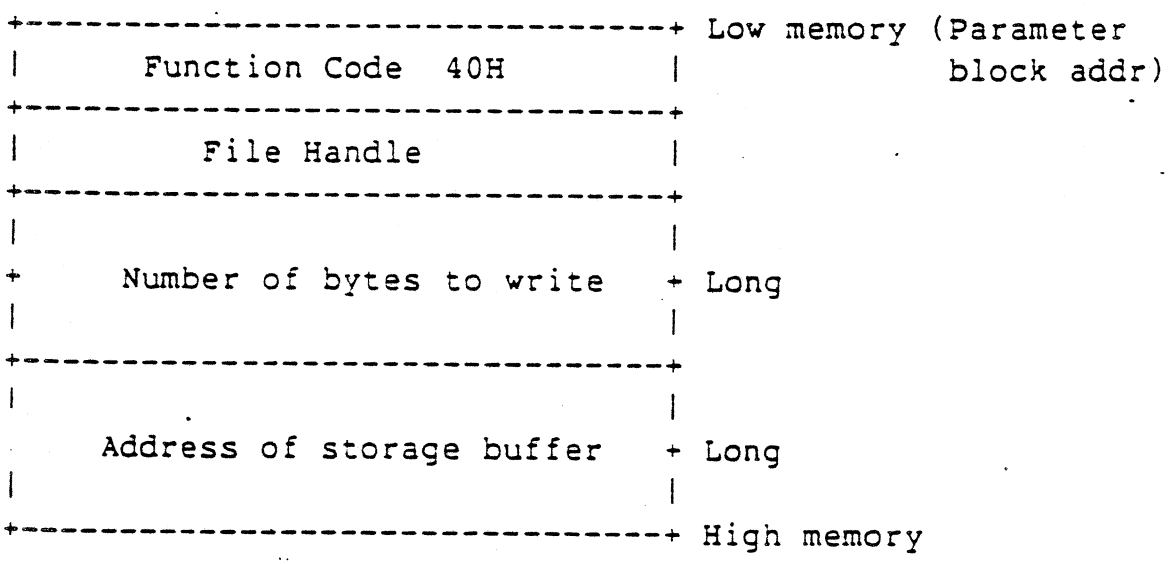
Register D0.L : Contains number of bytes read if read operation completed successfully, or an error code if an

error occurred. On an error condition, the high word of D0.L is set to FFFF.

38. 40: WRITE

```
  LONG
rite(handle,cnt,pbuffer)
  WORD      handle;
  LONG      cnt;
  BYTE      *pbuffer;
```

Parameter Block



rd contains the file handle.

ng 1 contains the number of bytes to write

ng 2 contains the buffer location containing the bytes to write.

Return Parameters:

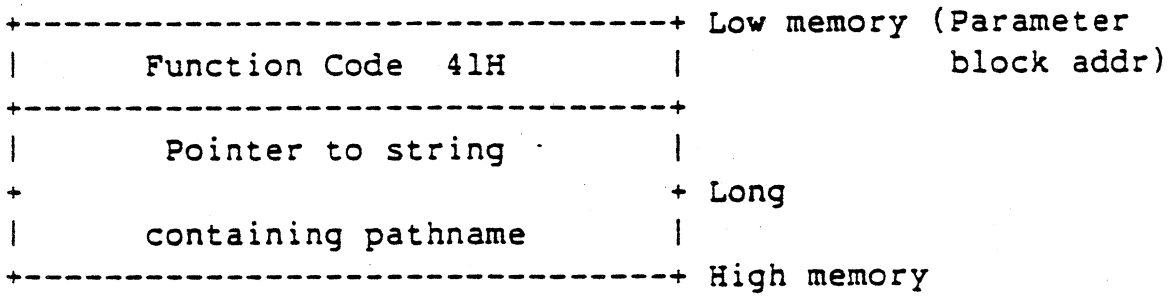
Register D0.L : Contains number of bytes written if write operation completed successfully, or an error code if an error occurred. On an error condition, the high word of D0.L is set to FFFF.

3.39. 41: UNLINK

```
   LONG
Unlink(name)
   BYTE      *name;
```

UNLINK deletes files.

Parameter Block



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

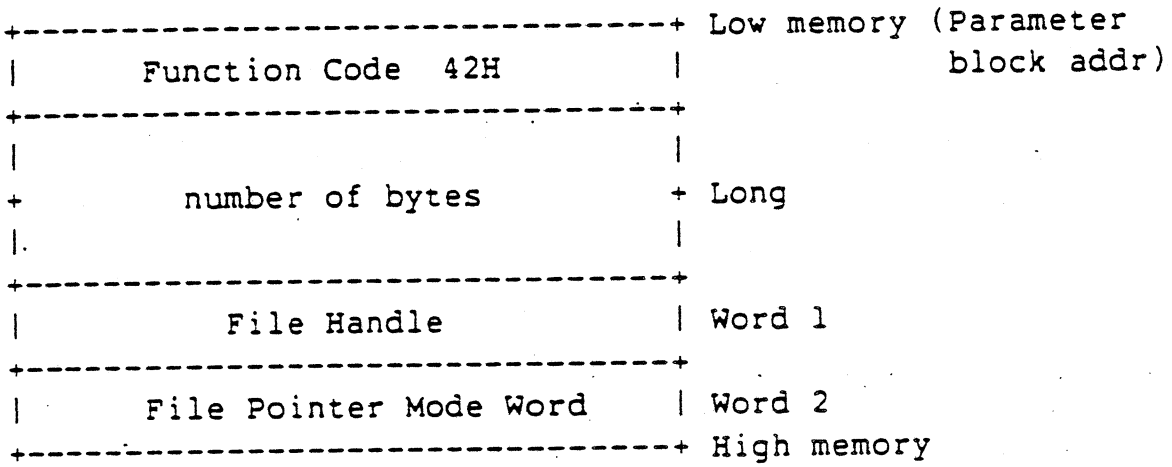
Return Parameters:

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

8.40. 42: LSEEK

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

Parameter Block



Long contains N, the number-of-bytes argument.

Word 1 is the file handle.

Word 2 is the method used to move the file pointer:

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

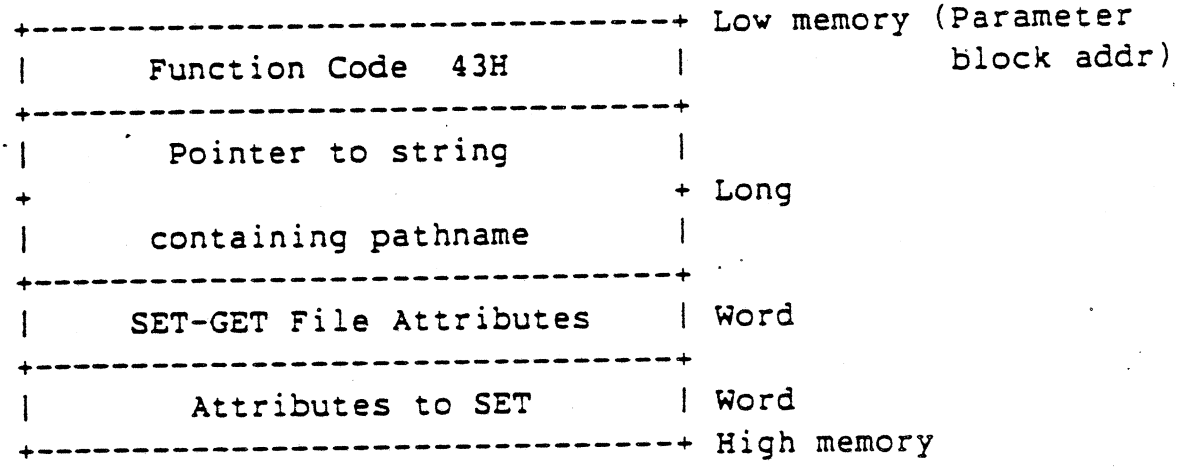
eturn Parameters:

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

8.41. 43: CHMOD

```
    LONG
chmod(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

turn Parameters:

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

8.42. 45: DUP

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

Input is a standard handle. Output is a non-standard handle number that points to the same file or device as the standard handle did.

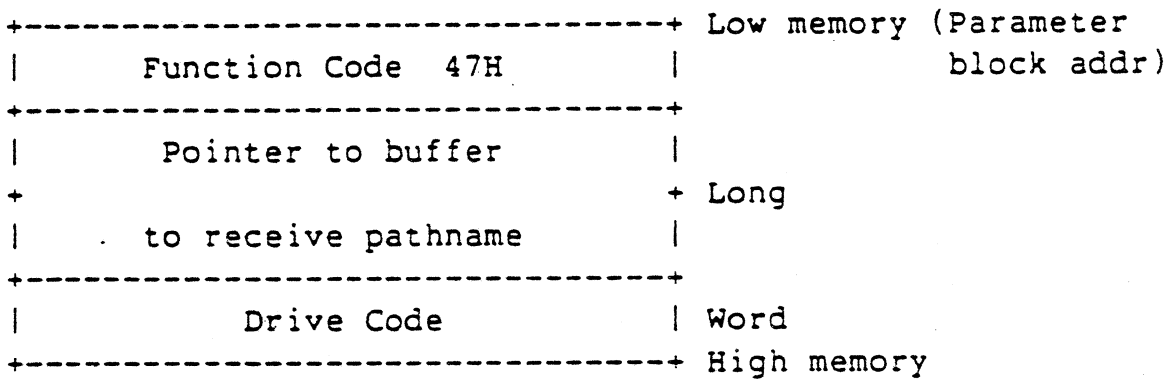
143. 46: FORCE
VOID force(stdhnd, nsthnd)

Forces the standard handle to point to the same file or device as the non-standard handle.

8.44. 47: GETDIR

```
    LONG
getdir(drvpath,drive)
    BYTE    *pdrvpath;
    WORD    drive;
{
}
```

Parameter Block



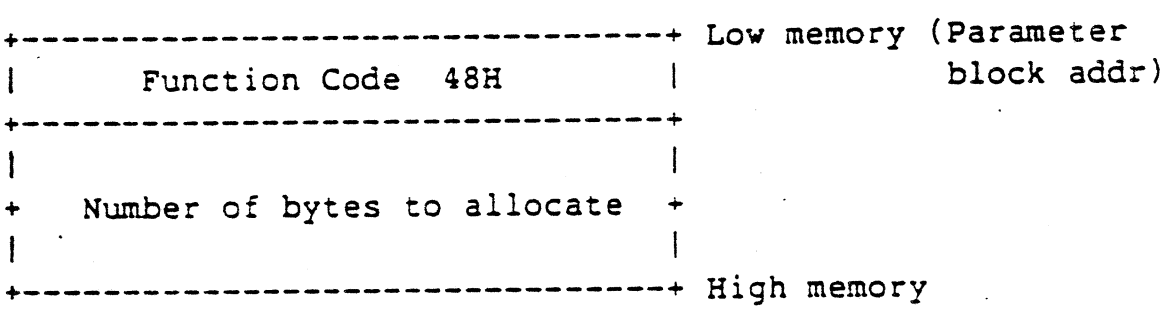
Long contains the address of a 64-byte buffer that receives the complete pathname of the current directory.

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

45. 48: MALLOC

```
  LONG
alloc(nbytes)
  LONG    nbytes;
```

Parameter Block



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

eturn Parameters:

egister 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.

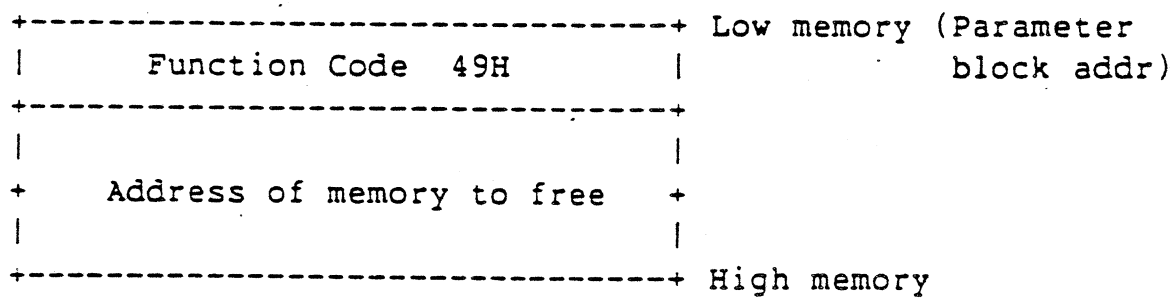
8.46. 49: MFREE

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

Entry parameters:

Register D0.W : 49H

Parameter Block



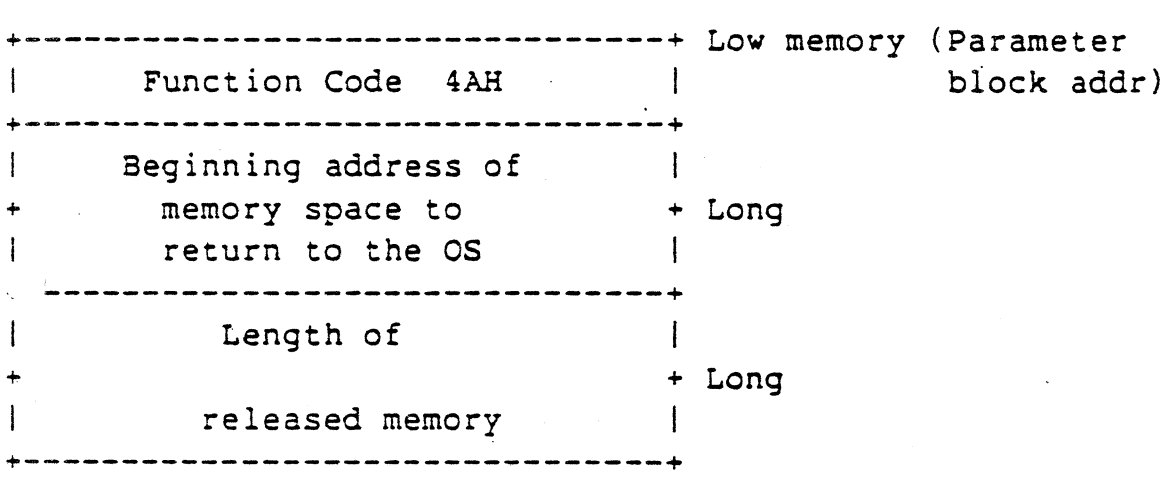
Long contains address of memory to free.

Return Parameters:

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

7. 4A: SETBLOCK

Parameter Block



nction:

he GEMDOS memory model, stack space grows downwards from high mory and program and data space grows upwards from low memory. e SETBLOCK function call polices memory space and reallocates used memory for GEMDOS's use. Long 1 contains the beginning dress of the memory to be returned to GEMDOS. Long 2 contains length of the returned space.

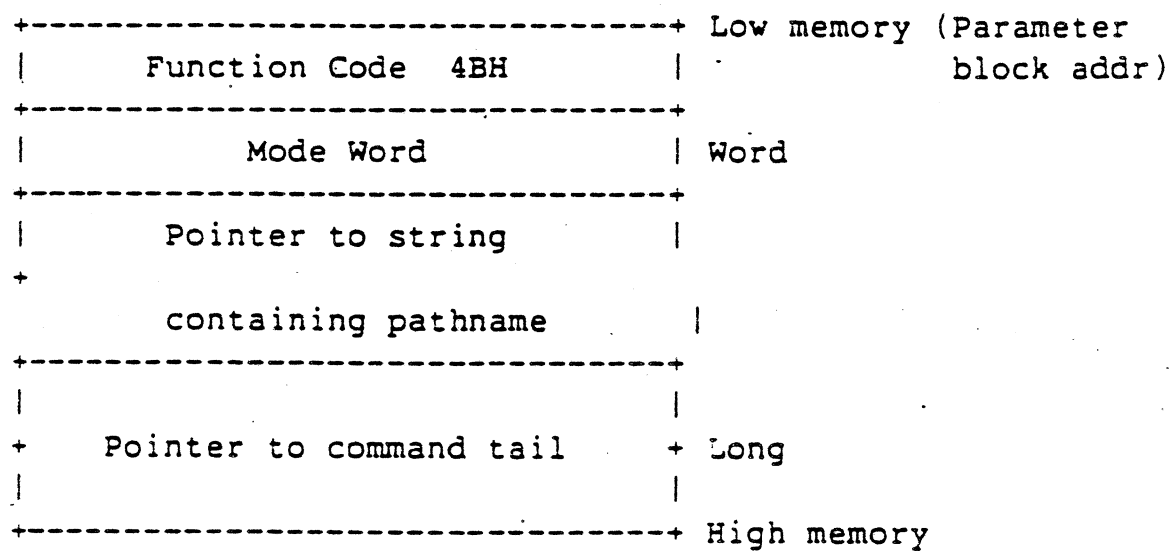
turn Parameters:

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

8.48. 4B: EXEC

```
    LONG
exec(load,pcspec,pcmdln)
    WORD      load;
    BYTE      pcspec;
    BYTE      pcdln;
{
}
```

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

ection details.

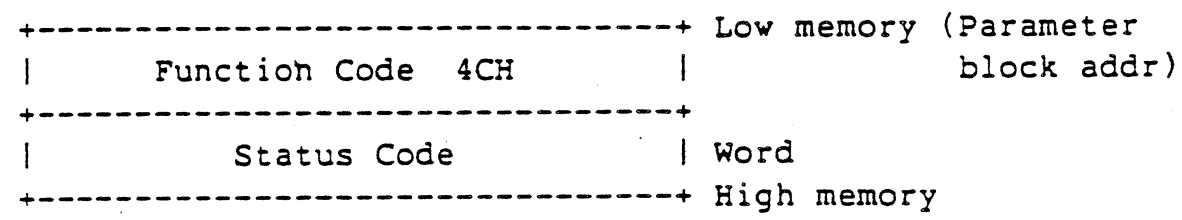
rn Parameters:

ster D0.L : Contains base page address.

8.49. 4C: TERM

```
    VOID
term()
{
}
```

Parameter Block



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

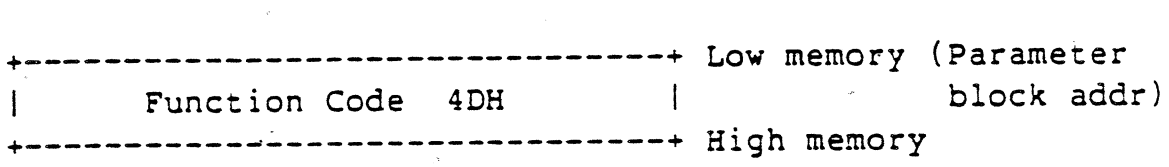
Register D0.L : Contains 0 if the file was terminated successfully, non-zero if an error occurred.

Function:

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

). 4D: Retrieve the Return Code of a Child

Parameter Block



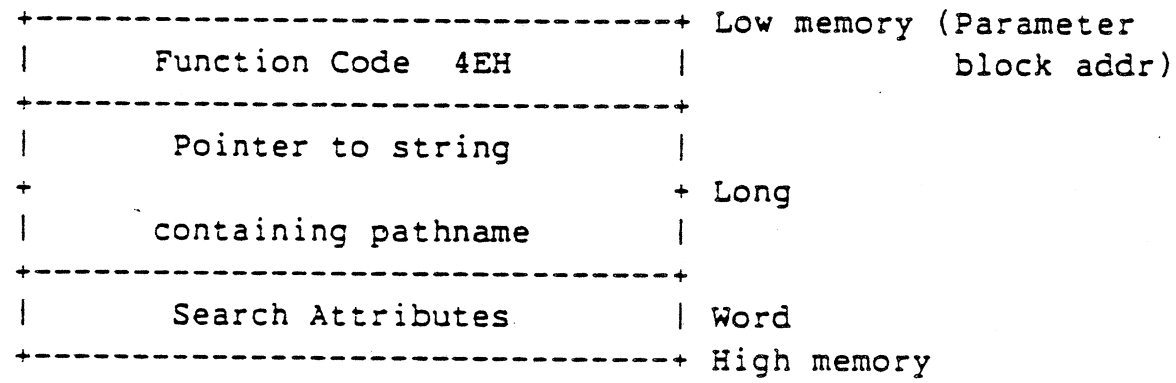
urn Parameters:

ister D0.W : Contains the return code of the child process.

8.51. 4E: SFIRST

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

Parameter Block



Long is a pointer to the null-terminated string specifying the file to find.

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

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

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

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

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

turn arameters:

gister D0.W : Contains 0 if no matching file was found;
1 if matching file was found.

nction:

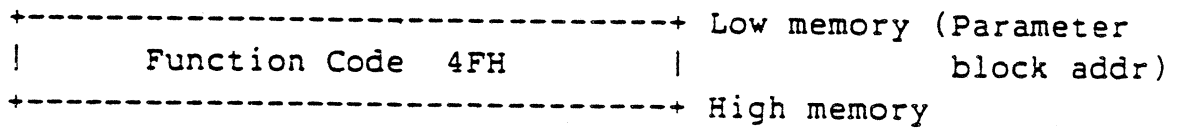
arches for a match for the specified filename, according to the attribute bit settings scribed above. If a match is found, a 44-byte DMA buffer is formatted as follows:

Longs	Contents
0	Reserved for OS use
21	File attributes
22 - 23	File time stamp
24 - 25	File date stamp
26 - 27	Low word of file size
28 - 29	High word of file size
30 - 43	Name and extension of found file

8.52. 4F: SNEXT

```
    LONG
snext()
{
}
```

Parameter Block



Return Parameters:

Register D0.W : Contains 0 if there are no more files to search;
1 if there are further files available.

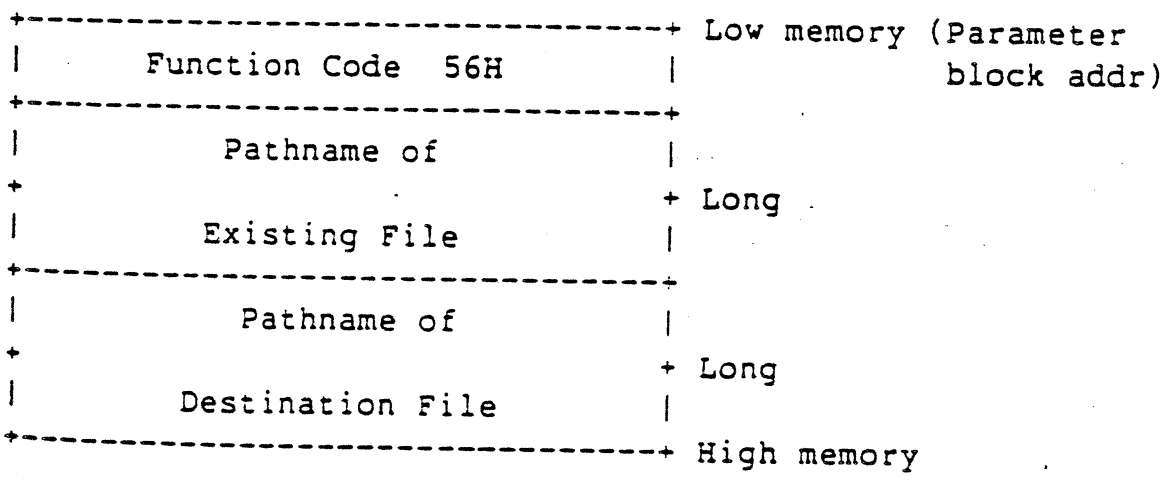
Function:

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.

53 56: RENAME

```
  LONG
name(n,p1,p2)
  WORD   n;
  BYTE   *p1;
  BYTE   *p2;
```

Parameter Block



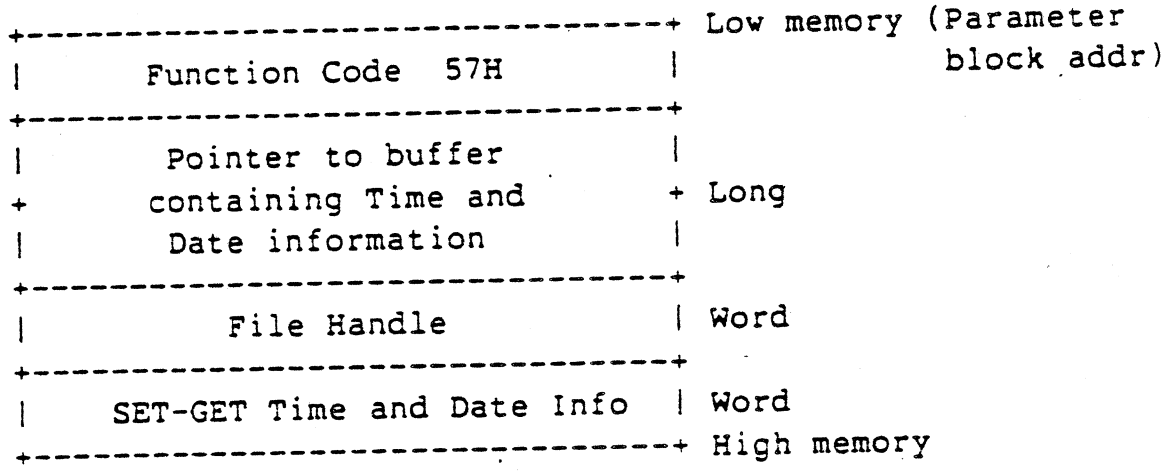
ng 1 is a pointer to the pathname of the existing file

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

8.54. 57: GSDTOF

```
    VOID
gsdtof(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

9 through 15 indicate the year (since 1980) in the range 0-119

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

9. BIOS Function Calls

9.1. 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
```

92 00

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

On 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;
  MD *mp_mal;
  MD *mp_rover;
} ;

MD /* memory descriptor */
   /* Only 1 allowed in this first release */
{
  MD *m_link;
  long m_start;
  long m_length;
  PD *m_own;
} ;
```

9.3. 01

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

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

0	PRT:
1	AUX:
2	CON:

Returns status in D0.L:

-1	device is ready
0	device is not ready

9.4. 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 IBM PC compatible scan code in the upper byte of the low 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.

9.5. 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.

. 04

```
NG read_write_sectors(wrtflg, buffer, num, recn, drive)
)RD wrtflg;
ır *buffer;
RD num;
RD recn;
RD drive;
```

wrtflg	is $0 for read, $1 for write
buffer	is a long pointer to a byte address for the disk transfer
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, ...

ction returns a 2's complement error number in D0.L. It is the responsibility of the er to check for media change before any write to FAT sectors. If media has changed, write should take place, just return with error code.

9.7. 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.

Function returns a long address that was the previous entry.

9.8. 06

LONG get_timer_ticks()
()

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

Why not make this microseconds per tick and eliminate parameter on tick trap?

9.9. 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.

08

G character_output_status(h)
D h;

n a character device handle described in Function 01.

ms status in D0.L:

-1 device is ready
0 device is not ready

9.11. Extended Functions

Function ??

Exchange mouse vector

Function ??

Exchange VBLANK vector

Timer Ticks

BIOS must trap into the BDOS on every timer tick. The BIOS
es in 1 parameter - the number of milliseconds since the last

Which trap?
In register or on stack?
See issue above on function 06.

10. GEM-GSX Function Calls

Table of Contents

Appendix A: Future System Extensions and Enhancements

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 print 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