```
Interrupt List, part 12 of 18
Copyright (c) 1989-1999,2000 Ralf Brown
--------W-2F1600-----------------------------
INT 2F - MS Windows - WINDOWS ENHANCED MODE INSTALLATION CHECK
  AX = 1600h
Return: AL = status
      00h neither Windows 3.x enhanced mode nor Windows/386 2.x running
      01h Windows/386 2.x running
      80h XMS version 1 driver installed (neither Windows 3.x enhanced
      mode nor Windows/386 2.x running) (obsolete--see note)
      FFh Windows/386 2.x running
  AL = anything else
      AL = Windows major version number >= 3
      AH = Windows minor version number
Notes:  INT 2F/AH=16h comprises an API for non-Windows programs (DOS device
    drivers, TSRs, and applications) to cooperate with multitasking
    Windows/386 2.x and Windows 3.x and higher enhanced mode.
  certain calls are also supported in the Microsoft 80286 DOS extender in
    Windows standard mode
  this function served as the installation check and AX=1610h served to
    get the driver entry point for XMS version 1, which is now obsolete.
    Use AX=4300h and AX=4310h instead
  Windows95 reports version 4.00, Windows95B reports version 4.03
SeeAlso: AX=160Ah,AX=1610h,AX=4300h,AX=4680h
Index:  installation check;XMS version 1
--------W-2F1602-----------------------------
INT 2F - MS Windows/386 2.x - GET API ENTRY POINT
  AX = 1602h
Return: ES:DI -> Windows/386 2.x API procedure entry point
Notes:  this interface is supported in Windows 3.x and Windows95 only for 2.x
    compatibility
  to get the current virtual machine (VM) ID in Windows/386 2.x:
      AX = 0000h
      ES:DI -> return address
      JUMP to address returned from INT 2F/AX=1602h
  After JUMP, at return address:
      BX = current VM ID.
SeeAlso: AX=C020h
--------W-2F1603-----------------------------
INT 2F C - MS Windows/386 - GET INSTANCE DATA
  AX = 1603h
```

```
Return: AX = 5248h ('RH') if supported
      DS:SI -> Windows/386 instance data (see #02630)
Notes:  reportedly supported by RM Nimbus MS-DOS 3.3 kernel
  this function is called by DOSMGR when AX=1607h/BX=0015h is not
    supported, as is the case in DOS versions prior to 5.0
  see Geoff Chappell's book _DOS_Internals_ for additional discussions of
    this function, DOSMGR's behavior, and instancing in general
SeeAlso: AX=1607h/BX=0015h

Format of Windows/386 instance data:
Offset  Size  Description (Table 02630)
 00h  WORD  segment of IO.SYS (0000h = default 0070h)
 02h  WORD  offset in IO.SYS of STACKS data structure (DOS 3.2x)
    0000h if not applicable
 04h  WORD  number of instance data entries (max 32)
 06h  Array of instance data entries
  Offset  Size  Description
   00h  WORD  segment (0002h = DOS kernel)
   02h  WORD  offset
   04h  WORD  size
--------W-2F1605-----------------------------
INT 2F C - MS Windows - WINDOWS ENHANCED MODE & 286 DOSX INIT BROADCAST
  AX = 1605h
  ES:BX = 0000h:0000h
  DS:SI = 0000h:0000h
  CX = 0000h
  DX = flags
      bit 0 = 0 if Windows enhanced-mode initialization
      bit 0 = 1 if Microsoft 286 DOS extender initialization
      bits 1-15 reserved (undefined)
  DI = version number (major in upper byte, minor in lower)
Return: CX = 0000h if okay for Windows to load
  CX = FFFFh (other registers unchanged) if Windows 3.0 in standard mode
  CX <> 0 if Windows should not load
  ES:BX -> startup info structure (see #02631)
  DS:SI -> virtual86 mode enable/disable callback or 0000h:0000h
        (see #02634)
Notes:  the Windows enhanced mode loader and Microsoft 286 DOS extender will
    broadcast an INT 2F/AX=1605h call when initializing.  Any DOS device
    driver or TSR can watch for this broadcast and return the appropriate
    values.  If the driver or TSR returns CX <> 0, it is also its
```

```
    responsibility to display an error message (however, Windows95 is
    reported to load regardless of the returned CX).
  each handler must first chain to the prior INT 2F handler with
    registers unchanged before processing the call
  if the handler requires local data on a per-VM basis, it must store the
    returned ES:BX in the "next" field of a startup info structure and
    return a pointer to that structure in ES:BX
  a single TSR may set the V86 mode enable/disable callback; if DS:SI is
    already nonzero, the TSR must fail the initialization by setting CX
    nonzero
  MSD checks for Windows 3.0 running in standard mode by testing whether
    CX=FFFFh and other registers are unchanged on return
  Novell DOS v7.0 (Update 8 - Update 11) TASKMGR in multitasking mode
    uses this broadcast, even if TASKMGR.INI sets WinPresent= to OFF
  Microsoft's EMM386.EXE for DOS 5+ when installed with the NOEMS option
    changes its driver name from EMMQXXX0 to EMMXXXX0 while Windows is
    active
SeeAlso: AX=1606h,AX=1608h,AX=4B05h

Format of Windows Startup Information Structure:
Offset  Size  Description (Table 02631)
 00h  2 BYTEs major, minor version of info structure
 02h  DWORD pointer to next startup info structure or 0000h:0000h
 06h  DWORD pointer to ASCIZ name of virtual device file or 0000h:0000h
 0Ah  DWORD virtual device reference data (see #02633)
    (only used if above nonzero)
 0Eh  DWORD pointer to instance data records (see #02632) or 0000h:0000h
---structure version >= 4.0---
 12h  DWORD pointer to optionally-instanced data records (see #02632)
      or 0000h:0000h

Format of one Instance Item in array:
Offset  Size  Description (Table 02632)
 00h  DWORD address of instance data (end of array if 0000h:0000h)
 04h  WORD  size of instance data
SeeAlso: #02631

Format of Virtual Device Reference Data:
Offset  Size  Description (Table 02633)
 00h  DWORD physical address of ??? or 00000000h
 04h  DWORD physical address of ??? table
```

```
 08h  DWORD "DEST_PAGE" address to which pages must be mapped
 0Ch  N DWORDs  "SRC_PAGE" physical addresses of the pages
    00000000h = end of table
Note: EMM386.EXE sets the first pointer to the start of the device driver
    chain, the second pointer to a field of 40h bytes followed by a
    16-bit offset to the end of the SRC_PAGE table, and DEST_PAGE to
    the start segment of the UMB area
SeeAlso: #02631

(Table 02634)
Values Windows virtual mode enable/disable procedure is called with:
  AX = 0000h disable V86 mode
  AX = 0001h enable V86 mode
  interrupts disabled
Return: CF set on error
  CF clear if successful
  interrupts disabled
SeeAlso: #02631
--------W-2F1606-----------------------------
INT 2F C - MS Windows - WINDOWS ENHANCED MODE & 286 DOSX EXIT BROADCAST
  AX = 1606h
  DX = flags
      bit 0 = 0 if Windows enhanced-mode exit
      bit 0 = 1 if Microsoft 286 DOS extender exit
      bits 1-15 reserved (undefined)
Notes:  if the init broadcast fails (AX=1605h returned CX <> 0), then this
    broadcast will be issued immediately.
  this call will be issued in real mode
  Novell DOS v7.0 (Update 8 - Update 15) TASKMGR in multitasking mode
    uses this broadcast, even if TASKMGR.INI sets WinPresent= to OFF
SeeAlso: AX=1605h,AX=1609h
--------W-2F1607-----------------------------
INT 2F C - MS Windows - VIRTUAL DEVICE CALL OUT API
  AX = 1607h
  BX = virtual device ID (see #02642)
  CX = (usually) callout subfunction
Return: (usually) AX,BX,CX,DX,ES contain results
Notes:  more of a convention than an API, this call specifies a standard
    mechanism for Windows enhanced-mode virtual devices (VxD's) to talk
    to DOS device drivers and TSRs
  see below for details on several virtual devices
```

```
SeeAlso: AX=1605h,AX=1607h/BX=000Ch,AX=1607h/BX=0014h,AX=1607h/BX=0015h
SeeAlso: AX=1607h/BX=0018h,AX=1684h"DEVICE API",AX=C020h
--------W-2F1607BX0006-----------------------
INT 2F C - MS Windows - "V86MMGR" VIRTUAL DEVICE API
  AX = 1607h
  BX = 0006h (VxD identifier of "V86MMGR")
  CX = 0000h
Return: AX = status
      0000h if local A20 state changed
      1607h if A20 unchanged
      other if global A20 state changed
SeeAlso: AX=1607h"CALL OUT API"
--------W-2F1607BX000C-----------------------
INT 2F C - MS Windows - "VMD" VIRTUAL MOUSE DEVICE API
  AX = 1607h
  BX = 000Ch (VxD identifier of "VMD")
Return: CX = nonzero if mouse driver already virtualized
Note: VMD (Virtual Mouse Driver) calls this and then checks whether CX is
    nonzero; if yes, it will not automatically virtualize the mouse
    driver.  This would be used if MOUSE.COM already virtualizes
    itself using the Windows API.
SeeAlso: AX=1607h/BX=0014h,AX=1607h/BX=0015h
--------W-2F1607BX000D-----------------------
INT 2F C - MS Windows95 - "VKD" VIRTUAL DEVICE - ??? CALLOUT
  AX = 1607h
  BX = 000Dh (VxD ID for VKD)
  ???
Return: ???
SeeAlso: AX=1607h"CALL OUT API",#02642
--------W-2F1607BX0010-----------------------
INT 2F C - MS Windows 3.1 - "BLOCKDEV" VIRTUAL HARD DISK DEVICE API
  AX = 1607h
  BX = 0010h (VxD identifier of "BLOCKDEV")
  CX = function
      0001h starting FastDisk compatibility tests
      0002h ending FastDisk compatibility tests
      0003h check if FastDisk installation allowed
    Return: CX = 0000h if allowed
Note: this interface is called by the Windows FastDisk driver (such as
    WDCTRL) when it thinks that the INT 13h handler immediately below
    IO.SYS's INT 13h code is not in ROM; it should be supported by any
```

```
    program which hooks itself underneath IO.SYS's INT 13h code with
    INT 2F/AH=13h
SeeAlso: AX=1607h/BX=0014h,INT 2F/AH=13h
--------W-2F1607BX0014-----------------------
INT 2F C - MS Windows - "VNETBIOS" VIRTUAL DEVICE API
  AX = 1607h
  BX = 0014h (VxD identifier of "VNETBIOS")
Return: ES:DI -> 128-byte table specifying VNETBIOS actions for each NetBIOS
    command code (see #02635)
Note: VNETBIOS (Virtual NetBIOS) calls this function to determine whether
    the NetBIOS has an extensions Windows should know about
SeeAlso: AX=1607h/BX=000Ch,AX=1607h/BX=0010h,AX=1607h/BX=0015h

(Table 02635)
Values for VNETBIOS action code:
 00h  "VN_Unknown" unknown command
 04h  "VN_No_Map"  no memory mapping necessary
 08h  "VN_Map_In"  input buffer is quickly used, so no global mapping needed
 0Ch  "VN_Map_In"  output buffer is quickly used, so no global mapping needed
 10h  "VN_Map_In_Out"  buffer is quickly used, so no global mapping needed
 14h  "VN_Chain_Send"  the chain-send command
 18h  "VN_Cancel" special case for cancel command
 1Ch  "VN_Buffer_In"  buffer is incoming
 20h  "VN_Buffer_Out" buffer is outgoing
 24h  "VN_Buffer_In_Out" buffer used for both incoming and outgoing data
--------D-2F1607BX0015-----------------------
INT 2F C - MS Windows - "DOSMGR" VIRTUAL DEVICE API
  AX = 1607h
  BX = 0015h (VxD identifier of "DOSMGR")
  CX = function
      0000h query instance processing
    DX = 0000h
    Return: CX = state
          0000h not instanced
          other instanced (DOS 5+ kernel returns 0001h)
        DX = segment of DOS drivers or 0000h for
          default of 0070h
        ES:BX -> patch table (see #02637)
      0001h set patches in DOS
    DX = bit mask of patch requests (see #02636)
    Return: AX = B97Ch
```

```
      BX = bit mask of patches applied (see #02636)
      DX = A2ABh
      0002h remove patches in DOS (ignored by DOS 5.0 kernel)
    DX = bit mask of patch requests (see #02636)
    Return: CX = 0000h (DOS 5-6)
    Note: return values are ignored by DOSMGR in Windows 3.1
      0003h get size of DOS data structures
    DX = bit mask of request (only one bit can be set)
        bit 0: Current Directory Structure size
    Return: if supported request:
          AX = B97Ch
          CX = size in bytes of requested structure
          DX = A2ABh
      else:
          CX = 0000h
          all other registers preserved
      0004h determine instanced data structures
    Return: AX = B97Ch if supported
      DX = A2ABh if supported (DOS 5+ kernel returns 0000h)
      BX = bit mask of instanced items
          bit 0: CDS
          bit 1: SFT
          bit 2: device list
          bit 3: DOS swappable data area
      0005h get device driver size
    ES = segment of device driver
    Return: DX:AX = 0000h:0000h on error (not dev. driver segment)
      DX:AX = A2ABh:B97Ch if successful
          BX:CX = size of device driver in bytes
Notes:  DOSMGR (DOS Manager) will check whether the OEM DOS/BIOS data has
    been instanced via this API and will not perform its own default
    instancing of the normal DOS/BIOS data if so; if this API is not
    supported, DOSMGR will also try to access instancing data through
    INT 2F/AX=1603h
  these functions are supported by the DOS 5+ kernel; DOSMGR contains
    tables of instancing information for earlier versions of DOS
  see Geoff Chappell's book _DOS_Internals_ for additional discussions of
    DOSMGR's behavior and instancing in general
SeeAlso: AX=1603h,AX=1605h,AX=1607h/BX=000Ch,AX=1607h/BX=0014h
SeeAlso: AX=1684h"DEVICE API"
```

```
Bitfields for DOSMGR patch requests:
Bit(s)  Description (Table 02636)
 0  enable critical sections
 1  NOP setting/checking user ID
 2  turn INT 21/AH=3Fh on STDIN into polling loop
 3  trap stack fault in "SYSINIT" to WIN386
 4  BIOS patch to trap "Insert disk X:" to WIN386

Format of DOSMGR patch table:
Offset  Size  Description (Table 02637)
 00h  2 BYTEs DOS version (major, minor)
 02h  WORD  offset in DOS data segment of "SAVEDS"
 04h  WORD  offset in DOS data segment of "SAVEBX"
 06h  WORD  offset in DOS data segment of InDOS flag
 08h  WORD  offset in DOS data segment of User ID word
 0Ah  WORD  offset in DOS data segment of "CritPatch" table to enable
      critical section calls (see INT 2A/AH=80h)
 0Ch  WORD  (DOS 5+ only) offset in DOS data segment of "UMB_HEAD",
      containing segment of last MCB in conventional memory
--------W-2F1607BX0018-----------------------
INT 2F C - MS Windows - "VMPoll" VIRTUAL DEVICE - IDLE CALLOUT
  AX = 1607h
  BX = 0018h (VMPoll VxD ID) (see #02642)
  CX = 0000h
Return: AX = status
      0000h if timeslice used
      nonzero if timeslice not needed
Note: when VMPoll makes this callout, all virtual machines are idle, and any
    interested TSR can use the opportunity to perform background
    processing
SeeAlso: AX=1607h"CALL OUT API",AX=1689h
--------W-2F1607BX0021-------------------------------------
INT 2F C - MS Windows - "PageFile" VIRTUAL DEVICE - GET LOCK BYTE
  AX = 1607h
  BX = 0021h (PageFile VxD ID)
  CX = 0000h
Return: AX = status
      0000h success
    ES:DI -> cache lock byte in disk cacher
      other no disk cache or unsupported
Notes:  PageFile issues this call on real-mode initialization in order to allow
```

```
    disk caches to provide it with a byte which it can use to temporarily
    lock the disk cache; VMPOLL also issues this call, so it is made
    twice each time Windows starts up
  if this call fails, PageFile falls back to other techniques for locking
    the disk cache
SeeAlso: AX=1607h"CALL OUT API"
--------W-2F1607BX002D-----------------------
INT 2F C - MS Windows - "W32S" VIRTUAL DEVICE - ??? CALLOUT
  AX = 1607h
  BX = 002Dh (VxD ID for W32S)
  ???
Return: ???
SeeAlso: AX=1607h"CALL OUT API",#02642
--------W-2F1607BX0040-----------------------
INT 2F C - MS Windows - "IFSMgr" VIRTUAL DEVICE - ??? CALLOUT
  AX = 1607h
  BX = 0040h (VxD ID for IFSMgr)
  ???
Return: ???
SeeAlso: AX=1607h"CALL OUT API",#02642
--------W-2F1607BX0446-----------------------
INT 2F C - MS Windows - "VADLIBD" VIRTUAL DEVICE - ??? CALLOUT
  AX = 1607h
  BX = 0446h (VxD ID for VADLIBD)
  ???
Return: ???
SeeAlso: AX=1607h"CALL OUT API",#02642
--------W-2F1607BX0484-----------------------
INT 2F C - MS Windows - "IFSMgr" VIRTUAL DEVICE - ??? CALLOUT
  AX = 1607h
  BX = 0484h (VxD ID for IFSMgr)
  ???
Return: ???
SeeAlso: AX=1607h"CALL OUT API",#02642
--------W-2F1607BX0487-----------------------
INT 2F C - MS Windows - "NWSUP" VIRTUAL DEVICE - ??? CALLOUT
  AX = 1607h
  BX = 0487h (VxD ID for NWSUP)
  ???
Return: ???
SeeAlso: AX=1607h"CALL OUT API",#02642
```

```
--------E-2F1607BX22C0-----------------------
INT 2F C - Rational Systems DOS/4GW - ???
  AX = 1607h
  BX = 22C0h
  ???
Return: ???
SeeAlso: INT 15/AX=BF02h,INT 15/AX=BF04h,#02642
--------W-2F1607BX28A1----------------------
INT 2F C - MS Windows - "PharLap" VIRTUAL DEVICE - ??? CALLOUT
  AX = 1607h
  BX = 28A1h (VxD ID for PharLap)
  ???
Return: ???
SeeAlso: AX=1607h"CALL OUT API",#02642
--------W-2F1607BX7A5F----------------------
INT 2F C - MS Windows - "SIWVID" VIRTUAL DEVICE - ??? CALLOUT
  AX = 1607h
  BX = 7A5Fh (VxD ID for SIWVID)
  ???
Return: ???
SeeAlso: AX=1607h"CALL OUT API",#02642
--------W-2F1608----------------------------
INT 2F C - MS Windows - WINDOWS ENHANCED MODE INIT COMPLETE BROADCAST
  AX = 1608h
Notes:  called after all installable devices have been initialized
  real-mode software may be called between the Windows enhanced-mode init
    call (AX=1605h) and this call; the software must detect this
    situation
SeeAlso: AX=1605h,AX=1609h
--------W-2F1609----------------------------
INT 2F C - MS Windows - WINDOWS ENHANCED MODE BEGIN EXIT BROADCAST
  AX = 1609h
Note: called at the beginning of a normal exit sequence; not made in the
    event of a fatal system crash
SeeAlso: AX=1606h,AX=1608h
--------W-2F160A----------------------------
INT 2F - MS Windows 3.1 - IDENTIFY WINDOWS VERSION AND TYPE
  AX = 160Ah
Return: AX = 0000h if call supported
      BX = version (BH=major, BL=minor)
      CX = mode (0002h = standard, 0003h = enhanced)
```

```
Note: Windows95 reports version 4.00, Windows95B reports version 4.03
SeeAlso: AX=1600h,AX=4680h
--------W-2F160B-----------------------------
INT 2F - MS Windows 3.1 - IDENTIFY TSRs
  AX = 160Bh
  ES:DI = 0000h:0000h
Return: ES:DI -> TSR information structure (see #02638)
Desc: this call allows Windows-aware TSRs to make themselves known to
    Windows.
Note: the TSR should first chain to the previous INT 2F handler, then
    allocate a communication structure, place the returned ES:DI
    pointer in the first field, and return a pointer to the new
    structure
SeeAlso: AX=1605h,AX=160Ch,AX=4B01h,AX=4B05h

Format of TSR-to-Windows information structure:
Offset  Size  Description (Table 02638)
 00h  DWORD pointer to next structure
 04h  WORD  PSP segment
 06h  WORD  API version ID (0100h)
 08h  WORD  EXEC flags (how to load command specified by "exec_cmd")
    bit 0: "WINEXEC"
    bit 1: "LOADLIBRARY"
    bit 2: "OPENDRIVER"
 0Ah  WORD  "exec_cmd_show" (see #02639)
 0Ch  DWORD "exec_cmd" pointer to command line to be executed
 10h  4 BYTEs reserved (0)
 14h  DWORD pointer to TSR ID block (see #02640)
 18h  DWORD pointer to TSR data block or 0000h:0000h

(Table 02639)
Values for TSR information structure "exec_cmd_show":
 00h  HIDE
 01h  SHOWNORMAL
 02h  SHOWMINIMIZED
 03h  SHOWMAXIMIZED
 04h  SHOWNOACTIVE
 05h  SHOW
 06h  MINIMIZE
 07h  SHOWMINNOACTIVE
 08h  SHOWNA
```

```
 09h  RESTORE
Note: this value is passed as the second parameter to the WinExec(),
    LoadLibrary(), or OpenDriver() call used to execute a requested
    command line
SeeAlso: #02638

Format of Norton Utilities 6.0 TSR ID block:
Offset  Size  Description (Table 02640)
 00h  WORD  length of name string
 02h  N BYTEs name of TSR's executable
SeeAlso: #02638
--------W-2F160C-----------------------------
INT 2F - MS Windows 3.1 - DETECT ROMs
  AX = 160Ch
  ???
Return: ???
Note: used by ROM Windows; appears to be a NOP under standard Windows95 and
    Windows95B
SeeAlso: AX=160Bh,INT 21/AH=6Dh"ROM"
--------D-2F160E-----------------------------
INT 2F U - MS-DOS 7 kernel - BOOT LOGO SUPPORT???
  AX = 160Eh
  BL = subfunction
      00h get ???
    AX = state of flag manipulated by subfn 04h and 05h
        0000h clear
        FFFFh set
    DX = ??? (0000h)
      01h link in INT 10h??? handlers
      02h unlink INT 10h??? handlers
      03h ???
      04h set ??? flag
      05h clear ??? flag
Return: AX = 0000h if supported
      ???
SeeAlso: AX=160Fh,AX=1611h,AX=1614h
--------D-2F160F-----------------------------
INT 2F U - MS-DOS 7 kernel - GET/SET ??? HANDLER
  AX = 160Fh
  BL = subfunction
      00h get ??? handler
```

```
    Return: AX = 0000h if supported
          CX:DX -> handler to which control is passed after
              ??? executes
         = 160Fh inside a Windows 95B DOS box
      01h set ??? handler
    CX:DX -> new handler for ???
    Return: AX = 0000h if supported
Notes:  this function is not supported if ??? in the IO.SYS drivers portion of
    the kernel is an IRET instruction (as is the case on my system)
    rather than a FAR JMP
  the indicated handler seems to be related to INT 10 processing
SeeAlso: AX=160Eh,AX=1611h,AX=1614h
--------m-2F1610-----------------------------
INT 2F - XMS v1.x only - GET DRIVER ADDRESS
  AX = 1610h
  details unavailable
Note: this function and AX=1600h were only used in XMS version 1 and are now
    obsolete.  Use AX=4300h and AX=4310h instead
SeeAlso: AX=1600h,AX=4310h
--------D-2F1611-----------------------------
INT 2F U - MS-DOS 7 kernel - GET SHELL PARAMETERS
  AX = 1611h
Return: AX = 0000h if supported
      DS:DX -> primary shell's executable name
      DS:SI -> prinary shell command line (counted string)
      BH = ??? (00h)
      BL = ??? (00h,40h)
Desc: return the program name and commandline from the CONFIG.SYS SHELL=
    statement
SeeAlso: AX=160Eh,AX=160Fh,AX=1612h,AX=4A33h
--------D-2F1612-----------------------------
INT 2F U - MS-DOS 7 kernel - GET ???
  AX = 1612h
Return: AX = 0000h if supported
      ES:BX -> DOS 7 kernel data (see #02641)
Note: called by VTD.VXD; one of the returned data items is a pointer to the
    WORD in which the default CLOCK$ driver maintains its count of days
    since 01jan1980
SeeAlso: AX=160Fh,AX=1611h,AX=1613h

Format of MS-DOS 7.x ??? kernel data:
```

```
Offset  Size  Description (Table 02641)
 00h  WORD  structure revision??? (0001h)
 02h  DWORD -> ??? function (call with DS=high word of this field)
    the indicated function vectors through the INT 13 hook at
      0070h:00B4h and then forces the A20 gate open
 06h  DWORD -> ??? function
 0Ah  WORD  DOS DS
 0Ch  8 BYTEs zeros seen
 14h  DWORD -> ??? data
 18h  DWORD -> ??? data
  ???
--------D-2F1613-----------------------------
INT 2F - MS-DOS 7 kernel - GET SYSTEM.DAT (REGISTRY FILE) PATHNAME
  AX = 1613h
  ES:DI -> buffer for full ASCIZ pathname to Windows95 SYSTEM.DAT
  CX = buffer size in bytes
Return: AX = 0000h if supported
      ES:DI buffer filled
      CX = number of bytes copied into buffer
SeeAlso: AX=160Eh,AX=1611h,AX=1612h,AX=1614h,AX=1690h
--------D-2F1614-----------------------------
INT 2F U - MS-DOS 7 kernel - SET SYSTEM.DAT (REGISTRY FILE) PATHNAME
  AX = 1614h
  ES:DI -> ASCIZ pathname to Windows95 SYSTEM.DAT
Return: AX = status
      0000h if successful
      1614h not supported
      other: maximum length of pathname (004Eh for v4.00.950)
SeeAlso: AX=160Eh,AX=1611h,AX=1613h,AX=1690h
----------2F1615-----------------------------
INT 2F - Windows95 - SAVE32.COM - INSTALLATION CHECK
  AX = 1615h
Return: AX = 0000h if installed
      BX = segment of resident code
Program: SAVE32.COM is a TSR included in the Windows95 distribution which
    preserves the contents of 32-bit registers across invocations of
    all of the hardware interrupt handlers (which, for some older BIOSes
    and TSRs, do not properly preserve the high words of the 32-bit
    registers)
--------W-2F1680-----------------------------
INT 2F - MS Windows, DPMI, various - RELEASE CURRENT VIRTUAL MACHINE TIME-SLICE
```

```
  AX = 1680h
Return: AL = status
      00h if the call is supported
      80h (unchanged) if the call is not supported
Notes:  programs can use this function in idle loops to enhance performance
    under multitaskers; this call is supported by MS Windows 3+, DOS 5+,
    DPMI 1.0+, and in OS/2 2.0+ for multitasking DOS applications
  does not block the program; it just gives up the remainder of the time
    slice
  should not be used by Windows-specific programs
  when called very often without intermediate screen output under
    MS Windows 3.x, the VM will go into an idle-state and will not
    receive the next slice before 8 seconds have elapsed. This time can
    be changed in SYSTEM.INI through "IdleVMWakeUpTime=<seconds>".
    Setting it to zero results in a long wait.
  this function has no effect under OS/2 2.10-4.0 if the DOS box has an
    "Idle Sensitivity" setting of 100
SeeAlso: AX=1689h,INT 15/AX=1000h,INT 15/AX=5305h,INT 21/AH=89h,INT 7A/BX=000Ah
--------W-2F1681-----------------------------
INT 2F - MS Windows 3+ - BEGIN CRITICAL SECTION
  AX = 1681h
Notes:  used to prevent a task switch from occurring
  should be followed by an INT 2F/AX=1682h call as soon as possible
  nested calls are allowed, and must be followed by an appropriate number
    of "end critical section" calls
  not supported in Windows/386 2.x. Get INDOS flag with INT 21/AH=34h and
    increment by hand.
SeeAlso: AX=1682h,INT 15/AX=101Bh,INT 21/AH=34h
--------W-2F1682-----------------------------
INT 2F - MS Windows 3+ - END CRITICAL SECTION
  AX = 1682h
Notes:  not supported in Windows/386 2.x.  Get InDOS flag with INT 21/AH=34h
    and decrement by hand, taking care not to decrement InDOS flag
    through zero
SeeAlso: AX=1681h,INT 15/AX=101Ch,INT 21/AH=34h
--------W-2F1683-----------------------------
INT 2F - MS Windows 3+ - GET CURRENT VIRTUAL MACHINE ID
  AX = 1683h
Return: BX = current virtual machine (VM) ID
Notes:  Windows itself currently runs in VM 1, but this can't be relied upon
  VM IDs are reused when VMs are destroyed
```

```
  an ID of 0 will never be returned
SeeAlso: AX=1684h"DEVICE API",AX=1685h,AX=168Bh
--------W-2F1684----------------------------
INT 2F - MS Windows - GET DEVICE API ENTRY POINT
  AX = 1684h
  BX = virtual device (VxD) ID (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point, or 0:0 if the VxD does not support an API
Note: some Windows enhanced-mode virtual devices provide services that
    applications can access.  For example, the Virtual Display Device
    (VDD) provides an API used in turn by WINOLDAP.
SeeAlso: AX=1684h/BX=0001h,AX=1684h/BX=0015h,AX=1683h,AX=4011h,INT 20"Windows"

(Table 02642)
Values for MS Windows VxD ID:
Value Name   CallOut V86 PM Description
 0000h  ACT200L      ?  ?  ?  IrDA Infrared ActiSys framer VxD
 0000h  ACT220L      ?  ?  ?  IrDA Infrared ActiSys 220 framer VxD
 0000h  ADAPTEC      ?  ?  ?  IrDA Infrared Adaptec framer VxD
 0000h  AM1500T      ?  N  N  (Win95)
 0000h  ATI      ?  N  N  (Win95) ATI display driver
 0000h  ATIPPCAP     ?  N  Y  ATI Rage128-based video card
 0000h  CDFS       ?  N  N
 0000h  CDTSD      ?  N  N  (Win95) CD-ROM Type-Specific Driver
 0000h  CE2NDIS3     ?  N  N  (W4Wg)
 0000h  CENDIS       ?  N  N  (W4Wg)
 0000h  CHIPS      ?  N  N  (Win95) Chips&Tech display driver
 0000h  CIRRUS       ?  N  N  (Win95) Cirrus display driver
 0000h  CTNDW      ?  N  N  (W4Wg)
 0000h  CTVSD      ?  N  N  (Win95) CD-ROM Vendor-Specific Driver
 0000h  CM2NDIS3     ?  N  N  (W4Wg)
 0000h  COMBUFF      ?  N  N  (Win95)
 0000h  COMPAQ       ?  N  N  (Win95) Compaq display driver
 0000h  CPQNDIS3     ?  N  N  (W4Wg)
 0000h  CRYSTAL      ?  ?  ?  IrDA Infrared Crystal framer VxD
 0000h  CWCENUM      ?  N  N  (Win95SR2???)
 0000h  CWCSPUD3     ?  N  N  (Win95SR2???)
 0000h  DBKVSSD      ?  N  N  (Win95) Databook PCMCIA socket services???
 0000h  DDOM95       ?  N  N
 0000h  DECLAN       ?  N  N  (W4Wg)
 0000h  DiskTSD      ?  N  N  (Win95) hard-disk Type-Specific Driver
```

```
0000h  DiskVSD      ?  N  N  (Win95) hard-disk Vendor-Specific Driver
0000h  DMICTVXD     ?  N  N
0000h  DMMDVDX      ?  N  N  Diamond MaximumDVD
0000h  DRVSPACX     ?  N  N  (Win95)
0000h  E30N3      ?  N  N  (W4Wg)
0000h  E31N3   N  N  (W4Wg)
0000h  EE16    N  N  (W4Wg)
0000h  EISA    N  N  (Win95)
0000h  EL59X   N  N  (Win95)
0000h  ELNK16    N  N  (W4Wg)
0000h  ELNK3   N  N  (Win95)
0000h  ELNKII    N  N  (W4Wg)
0000h  ELNKMC    N  N  (W4Wg)
0000h  ELPC3   N  N  (W4Wg)
0000h  ENABLE2   N  N  (Win95)
0000h  ENABLE4   N  N  (Win95)
0000h  EPRO    N  N  (Win95)
0000h  ES1488V   N  N  (Win95)
0000h  ES1688V   N  N  (Win95)
0000h  ES488V    N  N  (Win95)
0000h  ES688V    N  N  (Win95)
0000h  ESI   ?  ?  IrDA Infrared ESI framer VxD
0000h  FILEMON   N  N  DOS386 File Monitor
0000h  FLS1MTD   N  N  (Win95) flash-memory driver???
0000h  FLS2MTD   N  N  (Win95) flash-memory driver???
0000h  HPEISA    N  N  (W4Wg)
0000h  HPFEND    N  N  (W4Wg)
0000h  HPISA   N  N  (W4Wg)
0000h  HPMCA   N  N  (W4Wg)
0000h  HSFLOP    N  N
0000h  IBMTOK    N  N  (W4Wg)
0000h  IBMTOK4   N  N  (Win95)
0000h  IRCOMM    ?  ?  IrDA Infrared Virtual COM/LPT driver
0000h  IRLAMPEX  ?  ?  IrDA Infrared Protocol VxD
0000h  IRLAPFRM  ?  ?  IrDA Infrared Virtual COM/LPT frame driver
0000h  IRMATRAK  N  N  (W4Wg)
0000h  JAVASUP   N  N  Internet Explorer JAVA support
0000h  KEYREMAP  N  N  (Windows95 PowerToys) shift-key remapper
0000h  LPT      N  N  N  (Win4Workgroups 3.11) DOS386 LPT Device
0000h  LPTENUM      ?  N  N
0000h  MONVSD       ?  ?  ?
```

```
0000h  MGA      ?  N  N  (Win95) Matrox MGA display driver
0000h  MSMINI      ?  N  N  (Win95)
0000h  MSODISUP    N  N  N  (Win4Workgroups 3.11) MS ODI Support
0000h  mvpas     ?  N  N  (Win95) Pro Audio Spectrum driver
0000h  NECATAPI    ?  N  N  (Win95)
0000h  NICE      ?  N  N  (Win95)
0000h  NTI4CDR     ?  Y  Y  NTI CD-R/CD-RW
0000h  NV3      ?  N  N  (Win95SR2)
0000h  NWNBLINK    N  N  N  (Win4Workgroups 3.11) Netware NetBIOS
0000h  OAK   N  N  (Win95) Oak Tech display driver
0000h  OCTK32    N  N  (W4Wg)
0000h  OTCETH    N  N  (W4Wg)
0000h  PARALINK  N  N  (Win95)
0000h  PARALLAX  ?  ?  IrDA Infrared Parallax framer VxD
0000h  PCNTN3    N  N  (W4Wg)
0000h  PE3NDIS   N  N  (W4Wg)
0000h  PPM   N  N  (Win95)
0000h  PROTEON   N  N  (W4Wg)
0000h  QEMMFix   N  N
0000h  QIC117    N  N  (Win95) QIC-117 floppy-ctrl tape drive
0000h  QPI   N  N  QEMM Programming Interface (see INT 67/AH=3Fh)
0000h  RMM   N  N  Real-Mode Mapper for hw with real-mode drivers
0000h  S3    N  N  (Win95) S3 display driver
0000h  S3INFO    N  N
0000h  S3MINI    N  N  S3 display driver
0000h  SAGE    N  N  (Plus!) System Agent
0000h  sage    N  N  (Plus! for Win95) System Agent
0000h  scsi1hlp  N  N  (Win95)
0000h  SERENUM   N  N
0000h  SERIAL      N  N  N  (Win4Workgroups 3.11) DOS386 Serial Device
0000h  SERWAVE     ?  N  N
0000h  SETP3     ?  N  N  (Win95) Silicon Ethernet Pocket Adapter
0000h  SLMSDENM    ?  Y  Y
0000h  SMARTVSD    ?  N  N  (EZ-SMART???)
0000h  SMC8000W    ?  N  N  (W4Wg)
0000h  SMC80PC     ?  N  N  (W4Wg)
0000h  SMC8100W    ?  N  N  (W4Wg)
0000h  SMC8232W    ?  N  N  (W4Wg)
0000h  SMC9000     ?  N  N  (W4Wg)
0000h  SNIP      ?  N  N  (W4Wg)
0000h  SOCKET      ?  N  N  (W4Wg)
```

```
0000h  SOCKETSV     ?  N  N  (Win95)
0000h  SPAP       ?  Y  Y  (Win95)
0000h  SPENDIS      ?  N  N  (Win95)
0000h  SRAMMTD      ?  N  N  (Win95) flash-memory driver???
0000h  STLTH64      ?  N  N  Diamond Stealth64 driver
0000h  STLTHMON     ?  N  N
0000h  T20N3      ?  N  N  (W4Wg)
0000h  T30N3      ?  N  N  (W4Wg)
0000h  TCTOK      ?  N  N  (W4Wg)
0000h  TSENG      ?  N  N  (Win95) Tseng Labs display driver
0000h  UBNEI      ?  N  N  (W4Wg)
0000h  UNIMODEM     ?  ?  ?  (Win95) Universal Modem Driver
0000h  VDEF       ?  N  N  (Win95)
0000h  VGATEWAY     ?  N  Y  (Win95) dialin gateway
0000h  VIDEO7       ?  N  N  (Win95) Video7 display driver
0000h  VRomD      ?  N  N  (Win95)
0000h  VStDspcD     ?  ?  ?  Quarterdeck Stealth D*Space
0000h  VXDMON       ?  ?  ?
0000h  WD       ?  N  N  (Win95)
0000h  WINTOP       ?  N  N  (Windows95 Power Toys)
0000h  WSHTCP       ?  N  N
0000h  XGA      ?  N  N  (Win95) XGA display driver
0001h  VMM      ?  N  N  Virtual Machine Manager
0001h  VMM      ?  Y  Y  Windows95 Virtual Machine Manager
0002h  Debug      ?  ?  ?
0003h  VPICD      ?  Y  Y  Virtual Prog. Interrupt Controller (PIC) Device
0004h  VDMAD      ?  N  N  Virtual Direct Memory Access (DMA) Device
0005h  VTD      ?  Y  Y  Virtual Timer Device
0006h  V86MMGR      Y  N  N  (Windows3.x) Virtual 8086 Mode Device
0006h  V86MMGR      ?  N  Y  (Win95) Virtual 8068 Mode Device
0007h  PageSwap     ?  N  N  Paging Device
0008h  Parity       ?  N  N  Parity-check trapper
0009h  Reboot       ?  N  Y  Ctrl-Alt-Del handler
000Ah  VDD      ?  N  Y  Virtual Display Device (GRABBER)
000Bh  VSD      ?  N  N  Virtual Sound Device
000Ch  VMD      Y  Y  Y  Virtual Mouse Device
000Dh  VKD      ?  N  Y  Virtual Keyboard Device
000Eh  VCD      ?  N  Y  Virtual COMM Device
000Fh  VPD      ?  N  Y  Virtual Printer Device
0010h  VHD      ?  ?  ?  Virtual Hard Disk Device (Windows 3.0)
0010h  BLOCKDEV     Y  N  N  Virtual Hard Disk Device (Windows 3.1)
```

```
0010h  IOS       N  N  N  (Win4Workgroups 3.11) DOS386 IOS Device
0010h  IOS       ?  Y  Y  Windows95 I/O Supervisor
0011h  VMCPD       ?  Y  Y  (Windows3.x) Virtual Math CoProcessor Device
0011h  VMCPD       ?  N  Y  (Win95) Virtual Math CoProcessor Device
0012h  EBIOS       ?  N  N  Reserve EBIOS page (e.g., on PS/2)
0013h  BIOSXLAT      ?  N  N  Map ROM BIOS API between prot & V86 mode
0014h  VNETBIOS      Y  N  N  Virtual NetBIOS Device
0015h  DOSMGR        Y  Y  N  DOS data instancing (see #02656)
0016h  WINLOAD       ?  ?  ?
0017h  SHELL       ?  N  Y  (Windows3)
0017h  SHELL       ?  Y  Y  (Win95)
0018h  VMPOLL        Y  N  N
0019h  VPROD       ?  ?  ?
001Ah  DOSNET        ?  N  N  assures network integrity across VMs
001Ah  VNETWARE      ?  Y  Y  Novell NetWare DOSNET replacement
001Bh  VFD       ?  N  N  Virtual Floppy Device
001Ch  VDD2        ?  ?  ?  Secondary display adapter
001Ch  LoadHi        ?  N  N  Netroom LoadHi Device (RMLODHI.VXD)
001Ch  LoadHi        ?  N  N  386MAX LoadHi Device (386MAX.VXD)
001Ch  LoadHi        ?  N  N  Win386 LoadHi Device (EMM386.EXE)
001Dh  WINDEBUG      ?  N  Y
001Dh  TDDebug       ?  N  Y
001Eh  TSRLoad       ?  ?  ?  TSR instance utility
001Fh  BiosHook      ?  ?  ?  BIOS interrupt hooker VxD
0020h  Int13       N  N  N
0021h  PageFile      Y  N  Y  Paging File device
0022h  SCSI        ?  ?  ?
0022h  APIX        ?  N  Y  (Win95)
0023h  MCA_POS       ?  ?  ?  Microchannel Programmable Option Select
0024h  SCSIFD        ?  ?  ?  SCSI FastDisk device
0025h  VPEND       ?  ?  ?  Pen device
0026h  APM       ?  ?  ?  Advanced Power Management
0026h  VPOWERD       ?  Y  Y  (Win95) power management
0027h  VXDLDR        N  Y  Y  (Win4Wg 3.11/Win95) VXD Loader
0028h  NDIS        N  Y  Y  (Win4Wg 3.11) Network Driver Interface Spec
0029h  ???
002Ah  VWIN32        ?  N  Y  (Win95)
002Bh  VCOMM       N  Y  Y  (Win4Workgroups 3.11) DOS386 VCOMM Device
002Ch  SPOOLER       ?  N  N  Windows95 print spooler
002Dh  W32S        Y  N  Y  WIN32s 32-bit extension to Windows API
002Eh  ???
```

```
002Fh  ???
0030h  MACH32       N  N  Y  ATI Mach32 video card
0031h  NETBEUI      N  N  N  (Win4Workgroups 3.11) NETBEUI
0032h  SERVER       N  Y  Y  (Win4Workgroups 3.11) Int21 File Server
0032h  VSERVER      ?  N  Y  (Win95) Int21 File Server
0033h  CONFIGMG     ?  Y  Y  (Win95)
0033h  EDOS       ?  N  N  Windows DOS Box Enhancer by Mom's Software
0034h  DWCFGMG.SYS  ?  Y  ?  DOS Plug-and-Play configuration manager
0035h  SCSIPORT     ?  N  N  (Win95) virtualized access to SCSI adapter
0036h  VFBACKUP     ?  Y  Y  (Win95)
0037h  ENABLE       ?  Y  Y  (Win95)
0038h  VCOND      ?  Y  Y  (Win95)
0039h  ???
003Ah  VPMTD      N  N  Y  (Win4Workgroups 3.11) IFAX Scheduler Device
003Bh  DSVXD      ?  Y  N  DoubleSpace VxD from MS-DOS v6.x
003Ch  ISAPNP       ?  N  N  (Win95)
003Dh  BIOS       ?  Y  Y  (Win95)
003Eh  WSOCK      ?  Y  Y  (Win95) WinSock
003Fh  WSIPX      ?  N  N  (Win95) IPX WinSock
0040h  IFSMGR       ?  N  N  (Win95)
0041h  VCDFSD       ?  N  N  (Win95) CD-ROM File System Driver (MSCDEX)
0042h  MRCI2      ?  N  N  (Win95) DriveSpace3
0043h  PCI      ?  N  N  (Win95)
0048h  PERF       ?  N  N  (Win95)
004Ah  MTRR       ?  N  N  (Win95SR2) PPro/P-II MTRR enumerator???
004Bh  NTKERN       ?  N  Y  (Win95SR2)
0051h  ISAPNP       ?  N  N  (Win95) ISA Plug-and-Play manager
008Dh  ESDI_506     ?  N  N  (Win95) MFM/RLL/ESDI disk driver
0090h  voltrack     ?  N  N  (Win95) Volume Tracker
00FDh  FAKEIDE      ?  N  N  (Chicago)
0102h  CV1      ?  N  N  Microsoft C/C++ 7.00+ CodeView for Windows
011Fh  VFLATD       ?  N  Y  (Win95)
0200h  VIPX       ?  Y  Y  NetWare Virtual IPX Driver
0200h  VTEMPD       ?  ?  ?  dummy template driver by Ray Patch
0201h  VNWLSERV     ?  N  N  NetWare Lite 1.1 Server (SERVER.EXE)
0202h  WINICE       ?  Y  Y  SoftICE/W
0202h  SICE       ?  Y  Y
0203h  VCLIENT      ?  N  Y  NetWare Lite 1.1+ Client
0205h  VCAFT      ?  N  N  Novell Virtual CAFT Driver (LANalyzer for Win)
0205h  BCW      ?  Y  Y  Nu-Mega Bounds Checker for Windows
0206h  VTXRX      ?  N  N  Novell Virtual TXRX Driver (LANalyzer for Win)
```

```
0207h  DPMS        N  Y  N  Novell DOS Protected Mode Services
0234h  VCOMMUTE      ?  Y  Y  PC Tools Commute
0442h  VTDAPI        ?  N  Y  MMSys Win386 VTAPI Device
0443h  ???
0444h  VADMAD        ?  ?  ?  Autoinitialize DMA (Windows 3.0)
0445h  VSBD        ?  Y  Y  WinResKit: Sound Blaster Device
0446h  VADLIBD       Y  Y  Y  MMSys Win386 AdLib Device (v3.x)
0447h  ???
0448h  SETULTRA      ?  ?  ?  Gravis UltraSound setup
0449h  vjoyd       ?  N  Y  (Win95) joystick
044Ah  mmdevldr      ?  Y  Y  (Win95)
044Bh  ???
044Ch  msmpu401      ?  N  N  (Win95) MPU-401 MIDI driver
044Ch  cwdmidi       ?  Y  Y  (Crystal???) MIDI driver
044Dh  msopl       ?  N  N  (Win95) OPL-3 (SoundBlaster FM) driver
044Eh  mssblst       ?  N  N  (Win95) SoundBlaster MIDI driver
045Dh  VflatD        ?  N  Y  dva.386, part of WIN32s
045Eh  ???
045Fh  mssndsys      ?  ?  ?  Microsoft Sound System audio driver
045Fh  azt16       ?  Y  Y  Aztec Sound Galaxy 16 audio driver
0460h  UNIMODEM      ?  N  Y  Universal Modem driver
0480h  VNetSup       N  Y  Y  (Win4Workgrps 3.11) Virtual Network Support
0481h  VRedir        N  N  N  (Win4Workgroups 3.11) Redirector File System
0481h  VREDIR        ?  N  N  (Win95) Redirector File System driver
0482h  VBrowse       ?  Y  Y  Win386 Virtual Browser
0482h  SNAPVXD       ?  Y  Y  (Win95)
0483h  VSHARE        ?  N  N  (Win4Workgroups) Virtual SHARE
0483h  VSHARE        ?  Y  Y  (Win95) Virtual SHARE
0484h  IFSMgr        Y  N  Y  (Win4Wg 3.11) Installable File System Manager
0485h  ???     ???
0486h  VFAT        N  Y  Y  (Win4Workgroups 3.11) Win386 HPFS Driver
0487h  NWLINK        ?  Y  Y  Win386 Virtual Packet Exchange Protocol
0487h  NWSUP       Y  N  N  NetWare Vnetbios shim
0488h  VTDI        ?  N  N  (Win95)
0489h  VIP       ?  Y  N  (Win95)
0489h  FTCVIP        ?  Y  Y  Frontier Technologies' VIP
048Ah  VTCP        ?  Y  ?
048Ah  MSTCP       ?  Y  N  (Win95) TCP stack
048Ah  FTCTDI        ?  Y  Y  Future Technologies' TCP stack
048Bh  VCache        N  Y  Y  (Win4Workgroups 3.11) Virtual File Cache
048Bh  VCACHE        ?  Y  Y  (Win95) disk cache
```

```
048Ch  ???     ???
048Dh  RASMAC       ?  Y  Y  enhanced mode Win4Workgroups RASMAC device
048Eh  NWREDIR      ?  Y  Y  (Win95)
048Fh  ???     ???
0490h  ???     ???
0491h  FILESEC      ?  ?  ?  (Win95) File Access Control Manager
0492h  NWSERVER     ?  ?  ?  (Win95)
0493h  SECPROV      ?  ?  ?  (Win95) Security Provider
0494h  NSCL       ?  Y  Y  (Win95)
0495h  AFVXD      ?  N  N  (Win95)
0496h  NDIS2SUP     ?  ?  ?  (W4Wg???) NDIS2 networking support
0497h  MSODISUP     ?  N  N  (W4Wg???)
0498h  Splitter     ?  N  N  (Win95)
0499h  PPPMAC       ?  Y  Y  (Win95)
049Ah  VDHCP      ?  Y  Y  (Win95)
049Bh  VNBT       ?  Y  Y  (Win95) NetBIOS-over-TCP/IP driver
049Ch  ???
049Dh  LOGGER       ?  ?  ?  (Win95)
04A2h  IRLAMP       ?  ?  ?  IrDA Infrared Enumerator VxD
097Ch  PCCARD       ?  N  Y  (Win95) (see INT 20/VxD=097Ch)
1020h  VCV      ?  ?  ?  Microsoft C/C++ 7.00 CodeView
1021h  VMB      ?  Y  Y  Microsoft C/C++ 7.00 WXSRVR
1022h  Vpfd       ?  Y  Y  Microsoft C/C++ 7.00
1025h  MMD      ?  Y  Y  Microsoft C/C++ 8.00, Visual C/C++ 1.00
2020h  PIPE       ?  Y  Y  by Thomas W. Olson, in Windows/DOS DevJrn 5/92
21EAh  VADLIBWD     ?  N  Y  Adlib Waveform Driver by John Ridges
2200h  VFINTD       ?  Y  Y  Norton VFINTD (Norton Desktop)
22C0h  ???      Y    Rational Systems DOS/4GW ???
2402h  ZMAX       ?  N  N  Qualitas 386MAX v7 DOSMAX handler
24A0h  VNSS       ?  N  Y  Norton Screen Saver (Norton Desktop)
24A1h  VNDWD      ?  Y  Y  Norton VNDWD Device (Norton Desktop)
24A2h  SYMEvent     ?  Y  Y  Norton Utilities v8
2540h  VILD       ?  Y  N  INTERLNK client from MS-DOS v6.x
2640h  VASBID       ?  N  Y  WinResKit: Artisoft Sounding Board Device
2860h  COMMTASK     N  N  Y  Windows 386-mode preemptive tasker by James
         A. Kenemuth of Interabang Computing
28A0h  PHARLAPX     ?  Y  ?  PharLap inter-VM communications DLL
28A1h  PharLap      Y  Y  Y  PharLap 386|DOS-Extender DOSXNT.386
28C0h  VXD      N  Y  Y  Generic VxD for real and protected mode by
         Andrew Schulman in MSJ February 1993
28C1h  PUSHKEYS     ?  ?  ?  VKD_Force_Keys device
```

```
28C2h  VCR3D       ?  ?  ?  Virtual CR3, by A.Schulman in MSJ October 1992
2925h  EDOS        ?  Y  Y  Enhanced DOS by Firefly Software
292Dh  VSBPD       ?  Y  Y  Sound Blaster Pro
295Ah  GRVSULTR      ?  Y  Y  Gravis UltraSound / UltraSound ACE
3048h  FTCTCPIP      ?  N  Y  Frontier Technologies' TCP/IP stack
3049h  ???      (called by FNFSC32.VXD, FrontierTech's VNFSD)
304Ch  DWCFGMG.SYS  ?  Y Plug-and-Play configuration access
3098h  VstlthD       N  N  N  for QEMM Stealth ROM mode
3099h  VVidramD      ?  Y  N  for QEMM VIDRAM support
30F6h  WSVV        ?  N  Y  (Win95) WinSock for Voice-View Modems???
310Eh  WPS       ?  N  Y  MS DevNet CD-ROM: Windows Process Status
3110h  VGSSD       ?  Y  Y  VSGLX16.386 for Aztech Sound Galaxy 16
313Bh  PMC       ?  ?  ?  Power Management Coordinator
318Ah  LMOUSE        ?  Y  Y  (Win95) Logitech mouse???
31CFh  STAT.386      ?  ?  ?  Ton Plooy's processor statistics VxD
3202h  VdspD       ?  N  N  (Win95)
3203h  vpasd       ?  N  N  (Win95) Pro Audio Spectrum driver
32A4h  SBAWE       ?  Y  Y  (Win95) SoundBlaster AWE driver
32A5h  VSB16       ?  N  N  (Win95) SoundBlaster 16 driver
32CBh  VFRAD       ?  Y  Y  Dr.Franz - Simultan's diagnotics VFRAD.386
32DCh  NV3RM       ?  N  Y  (Win95SR2)
3354h  Discover      ?  N  Y  (Nuts&Bolts) Discover
33AAh  DECCORE       ?  Y  Y  (Win95) DEC Pathworks core VxD
33B4h  DECLICL       ?  N  N  (Win95)
33F0h  VIWD        ?  Y  Y  Gravis UltraSound Plug-n-Play Interwave v1.x
33FCh  ASPIENUM      ?  N  N  (Win95)
34DCh  MAGNARAM      ?  N  Y  Quarterdeck MagnaRAM (MAGNA31.VXD/MAGNA95.VXD)
357Eh  DSOUND        ?  Y  Y  (Win95) DirectSound
3584h  VSNDSYS       ?  Y  Y  (Win95SR2)
35C5h  LUGEPS        ?  Y  Y  Lugaru's Epsilon editor
36AEh  AIB-PC.386    ?  Y  Y  Sunset Laboratory interface hardware driver
377Bh  MX1501HAD     ?  ?  ?  Cherry keyboard chipcard reader
38BEh  Vheapx        ?  N  Y  (Nuts&Bolts) Virtual Heap Expander
38C0h  Bombshel      ?  N  Y  (Nuts&Bolts) Bombshelter
38DAh  VIWD        ?  Y  Y  UltraSound PnP InterWave driver v2.0beta
39E6h  A3D       ?  N  N  (Win95SR2)
3A39h  CINEMSYS      ?  Y  Y  Software Cinemaster MPEG/DVD decoder
3BFCh  CWCSPUD       ?  N  N  (Win95SR2)
3BFDh  CWCPROXY      ?  N  N  (Win95SR2)
3BFEh  CWCMMSYS      ?  N  Y  (Win95SR2)
3BFFh  CWCDSND       ?  N  N  (Win95SR2)
```

```
 3C46h  X10MOUSE     ?  Y  Y  X10 RF wireless mouse
 3C78h  VGARTD       ?  N  N  (Win95SR2)
 3E6Dh  DDRAW      ?  Y  Y  DirectDraw
 3ED6h  ATIVVXXX     ?  N  Y  ATI Rage128-based video card
 3EE5h  WINTEL.VXD   ?  ?  ?  "WinTel" Windows remote-control program
        (see also PORT 063Eh)
 4321h  POSTMSG      ?  Y  Y  (see #02712)
 4321h  VPCD       ?  N  N  PCache
 4321h  avvxp500     ?  N  N  (Win95) VxP500 driver
 6001h  REGVXD       ?  Y  Y  Windows95 Registry Monitor helper
 7A5Fh  SIWVID       Y  Y  Y  Soft-ICE for Windows video driver
 7FE0h  VSWITCHD     ?  Y  N  by Jeff Prosise
 7FE0h  VWFD       N  Y  Y  reports windowed/fullscreen state; by Neil
          Sandlin of Microsoft, shipped with ANSIPLUS
 7FE1h  VWATCHD      N  Y  Y  basic driver w/ no functionality except tracing
          by Keith Jin of Microsoft PSS
 7FE5h  VFINTD       N  Y  Y  Virtual Floppy Interrupt trapper by Neil
          Sandlin of Microsoft
 7FE7h  VMPAGES      N  Y  Y  demonstration of exporting VxD services, by
          Neil Sandlin of Microsoft
 7FE8h  VPOSTD       ?  Y  Y  PostMessage() demo by Curtis J. Palmer of MS
 7FE9h  VIdleD       N  N  N  demonstration of Call_When_Idle function, by
          Bernie McIlroy of Microsoft
 7FEBh  VMIOD      N  N  N  Virtual Monitor I/O Traffic Device, by Neil
          Sandlin of Microsoft
 7FEDh  VMIRQD       N  N  N  Virtual Monitor IRQ Traffic Device, by Neil
          Sandlin of Microsoft
 8888h  VbillD       ?  ?  ?  Bill Potvin II's for reversing Compaq LTE video
 EEEEh  VEPSD      ?  N  N  Virtual Extended Paging Services for
          Borland C++ v4.0
Notes:  The high bit of the VxD ID is reserved for future use. Originally,
    the next 10 bits were the OEM number which was assigned by Microsoft,
    and the low 5 bits were the device number.  Currently, Microsoft
    assigns VxD IDs individually for each driver; send blank email to
    vxdid@microsoft.com for more information.
  "CallOut"=Y indicates that the VxD uses the INT 2F/AX=1607h/BX=VxDID
    device callout interface; "PM" and "V86" indicate whether the VxD
    provides an API entry point in protected mode and Virtual-86 mode
    (e.g. DOS boxes)
--------W-2F1684BX0001-----------------------
INT 2F - MS Windows95 - VMM - GET API ENTRY POINT
```

```
  AX = 1684h
  BX = 0001h (virtual device ID for VMM) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02643)
      0000h:0000h if the VxD does not support an API
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02643)
Call Windows VMM 16-bit entry point with:
  AX = function number
      ---registry functions---
      0100h "RegOpenKey"
    STACK:  DWORD -> DWORD for returned key handle
      DWORD -> ASCIZ registry key name
      DWORD HKEY (see #02644)
      0101h "RegCreateKey"
    STACK:  DWORD -> DWORD for returned key handle
      DWORD -> ASCIZ registry key name
      DWORD HKEY (see #02644)
      0102h "RegCloseKey"
    STACK:  DWORD key handle from RegOpenKey or RegCreateKey
      0103h "RegDeleteKey"
    STACK:  DWORD -> ASCIZ registry key name
      DWORD HKEY (see #02644)
      0104h "RegSetValue"
    STACK:  DWORD ???
      DWORD -> ???
      DWORD ???
      DWORD -> ???
      DWORD HKEY (see #02644)
      0105h "RegQueryValue"
    STACK:  DWORD -> DWORD for ???
      DWORD -> ASCIZ ???
      DWORD -> ASCIZ ???
      DWORD HKEY (see #02644)
      0106h "RegEnumKey"
    STACK:  DWORD ???
      DWORD -> ASCIZ ???
      DWORD ???
      DWORD HKEY (see #02644)
      0107h "RegDeleteValue"
```

```
      0108h "RegEnumValue"
    STACK:  DWORD -> DWORD for ???
      DWORD -> BYTE ???
      DWORD -> DWORD for ???
      DWORD -> DWORD for ???
      DWORD -> DWORD for ???
      DWORD -> ASCIZ ???
      DWORD ???
      DWORD HKEY (see #02644)
      0109h "RegQueryValueEx"
      010Ah "RegSetValueEx"
      010Bh "RegFlushKey"
      010Ch "RegLoadKey"
      010Dh "RegUnLoadKey"
      010Eh "RegSaveKey"
      010Fh "RegRestore"
      0110h "RegRemapPreDefKey"
Return: parameters popped from stack
  DX:AX = return value

(Table 02644)
Values for Windows95 VMM predefined HKEY values:
 80000000h  HKEY_CLASSES_ROOT
 80000001h  HKEY_CURRENT_USER
 80000002h  HKEY_LOCAL_MACHINE
 80000003h  HKEY_USERS
 80000004h  HKEY_PERFORMANCE_DATA
 80000005h  HKEY_CURRENT_CONFIG
 80000006h  HKEY_DYN_DATA
SeeAlso: #02643
--------W-2F1684BX0003----------------------
INT 2F - MS Windows - VPICD - GET API ENTRY POINT
  AX = 1684h
  BX = 0003h (virtual device ID for VPICD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02645)
      0000h:0000h if the VxD does not support an API
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02645)
Call VPICD API entry point with:
```

```
  EAX = function number
      0000h get version
    Return: AX = binary version (AH=major, AL=minor)
      0001h virtualize timer???
      0002h unvirtualize timer???
--------W-2F1684BX0005-----------------------
INT 2F - MS Windows - VTD - GET API ENTRY POINT
  AX = 1684h
  BX = 0005h (virtual device ID for VTD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02646)
      0000h:0000h if the VxD does not support an API
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02646)
Call VTD.386/VTD.VXD entry point with:
  AX = function number
      0000h get VTD version number
    Return: CF clear
      AH = major version
      AL = minor version
      0100h get current clock tick time
    Return: EDX:EAX = clock tick time in 840ns units since Windows
          was started
      0101h get current system time in milliseconds
    Return: EAX = time in milliseconds that Windows has been
          running
      0102h get current virtual machine time
    Return: EAX = cumulative amount of time the virtual machine has
          been active, in milliseconds
Note: this entry point should only be called directly when TOOLHELP.DLL
    TimerCount() cannot be called
SeeAlso: #01268,#01270,#01269 at INT 20"Windows"
--------W-2F1684BX0006-----------------------
INT 2F P - MS Windows95 - V86MMGR - GET API ENTRY POINT
  AX = 1684h
  BX = 0006h (virtual device ID for V86MMGR device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02647)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
```

```
(Table 02647)
Call V86MMGR entry point with:
  EAX = function number
      0000h get V86MMGR version
    Return: CF clear
      AH = major version
      AL = minor version
      0001h get ???
    Return: CF clear
      EAX = status bits
        bit 0: ???
        bit 1: ???
        bit 2: ???
        bit 3: ???
        bit 4: ???
      else
    Return: CF set
--------W-2F1684BX0009-----------------------
INT 2F P - MS Windows - REBOOT - GET API ENTRY POINT
  AX = 1684h
  BX = 0009h (virtual device ID for REBOOT device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02648)
      0000h:0000h if the VxD does not support an API
SeeAlso: INT 14/AH=17h"FOSSIL",INT 16/AX=E0FFh

(Table 02648)
Call REBOOT protected-mode entry point with:
  AX = function
      0100h warm boot
    Return: never
    Note: broadcasts "Reboot_Processor" message, which is caught
        by the VKD device
      0201h set KERNEL Ctrl-Alt-Del handler
    ES:DI -> new Ctrl-Alt-Del handler
    DS:SI -> KERNEL reboot sanity check byte
    Return: CF clear
    Notes:  if an application installs its own handler and then
        chains to Windows' handler, Windows will no longer
        be able to detect hung applications, and will always
```

```
        produce an "Application not responding" dialog
      DS must contain a writable, fixed selector because
        the provided address is converted to a linear address
        before being stored
      when Ctrl-Alt-Del is pressed in the system VM, Reboot
        sets the sanity check byte to zero, schedules a
        750ms wait, and then tests whether the check byte is
        still zero; if not, it displays a message that there
        is no hung application and then exits
      0202h get KERNEL Ctrl-Alt-Del handler
    Return: CF clear
      ES:DI -> current Ctrl-Alt-Del handler
    Note: the default handler is located in KERNEL
      0203h display "Application not responding" dialog box
    ES:DI -> ASCIZ name of hung application
    Return: never if user pressed Ctrl-Alt-Del a second time
      CF clear
      AX = result
          0000h user pressed Esc
          0001h user pressed Enter
    Note: this function is used by the default Windows
        Ctrl-Alt-Del handler
      0204h set/reset protected-mode INT 01 handler
    CX:EDX -> new protected-mode INT 01 handler
    CX = 0000h restore protected-mode INT 01 handler
    Return: CF clear
    Notes:  if CX is nonzero, the current handler address is saved
        internally before the new handler is set; this saved
        address is then used when CX is zero on entry
      used by Windows' default Ctrl-Alt-Del handler; actual
        fatal exit to DOS will be done on next INT 01
    Warning: opened files are not closed and remain open as
        orphaned files in DOS
Note: functions 0201h and 0203h are not useful outside the system VM
SeeAlso: #01271,#01273
--------W-2F1684BX000A-----------------------
INT 2F P - MS Windows - VDD - GET API ENTRY POINT
  AX = 1684h
  BX = 000Ah (virtual device ID for VDD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02649)
```

```
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02649)
Call VDD entry point with:
  EAX = function
      0000h get VDD version
    Return: CF clear
      AH = major version
      AL = minor version
    Note: also performs an internal initialization
      0001h ???
    Return: ECX = ???
      ???
      0002h
      0003h
      0004h
      0005h
      0006h
      0007h
      0008h
      0009h
      0080h
      0081h
      0082h
      0083h
      0084h
      0085h
      0086h
      0087h
      0088h
      0089h
      else
    Return: nothing
--------W-2F1684BX000C-----------------------
INT 2F - MS Windows - VMD - GET API ENTRY POINT
  AX = 1684h
  BX = 000Ch (virtual device ID for VMD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02650)
      0000h:0000h if the VxD does not support API in current mode
```

```
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02650)
Call VMOUSE entry point with:
  EAX = function number
      0000h get VMOUSE version
    Return: CF clear
      AH = major version
      AL = minor version
      0001h
    EBX = ???
    ECX = ???
    Return: CF clear if successful
      CF set on error (e.g. fn 0003h not yet called)
      0002h ??? (calls "test system VM handle")
    Return: CF clear if successful (in system VM)
      CF set on error
      0003h ???
    ECX = ???
    DX = ???
    Return: CF clear
      0004h ???
    Note: invokes Call_Priority_VM_Event
      0005h get mouse port data
    Return: CF clear
      AL = ??? (04h)
      AH = mouse IRQ interrupt number (IRQ4=0Ch,etc.)
      CX = mouse I/O port address (e.g. 03F8h)
      DX = COM port number??? (0001h for mouse on COM1)
      0100h NOP???
    Return: CF clear
      0101h init???
    Return: CF clear
    Note: appears to be the same as fn 0005h, but returns no data
      0102h unimplemented
    Return: CF set
      0103h check ???
    Return: AX = status (0000h/0001h)
    Note: checks flag set by fn 0003h
      else
    Return: CF set
```

```
SeeAlso: #02649,#02651
--------W-2F1684BX000D-----------------------
INT 2F P - MS Windows - VKD - GET API ENTRY POINT
  AX = 1684h
  BX = 000Dh (virtual device ID for VKD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02651)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02651)
Call VKD entry point with:
  EAX = function
      0000h get VKD version
    Return: CF clear
      AH = major version
      AL = minor version
      0001h ???
    EBX = VM handle or 00000000h to use ??? VM handle
    CH = ???
    CL = ???
    EDX = ??? or FFFFFFFFh
    Return: CF clear if successful
      CF set on error
      else
    Return: CF set
SeeAlso: #02650,#02652
--------W-2F1684BX000E-----------------------
INT 2F P - MS Windows - VCD - GET API ENTRY POINT
  AX = 1684h
  BX = 000Eh (virtual device ID for VCD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02652)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02652)
Call VCD entry point with:
  EDX = function number
      0000h get VCD version
    Return: CF clear
```

```
      AH = major version
      AL = minor version
      0001h get ???
    Return: CF clear
      AX = bit mask of ???
      0002h get ???
    CX = COM port number
    Return: CF clear
      DX:AX -> ???
      0003h set ???
    CX = COM port number
    DX:AX -> new ???
    Return: CF clear
      0004h acquire COM port
    AX = ???
    CX = COM port number
    Return: CF clear
      AX = ???
      EBX = ???
      DX = ???
      0005h release COM port
    CX = COM port number
    Return: CF clear
      0006h ???
    Return: CF set
      AL = 00h
      else
    Return: CF set
      EAX = FFFFFFFFh
Note: these functions are apparently only available from the system VM,
    returning CF set and EAX=FFFFFFFFh otherwise
SeeAlso: #02651,#02653
--------W-2F1684BX000F-----------------------
INT 2F P - MS Windows - VPD - GET API ENTRY POINT
  AX = 1684h
  BX = 000Fh (virtual device ID for VPD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02653)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
```

```
(Table 02653)
Call VPD entry point with:
  EDX = function number
      0000h get VPD version
    Return: CF clear
      AH = major version
      AL = minor version
      DX = ??? (CB01h)
      0001h get valid??? printers
    Return: CF clear
      AX = bitmask of ??? printers (bits 0-2)
      0002h get ??? for printer
    CX = printer port (0-2)
    Return: CF clear if successful
          BX:AX = ???
      CF set on error (invalid port number)
      0003h set ??? for printer
    CX = printer port (0-2)
    BX:AX = ???
    Return: CF clear if successful
      CF set on error (invalid port number)
      0004h ???
    CX = printer port (0-2)
    EAX = VM handle
    Return: CF clear if successful
      CF set on error (invalid port number)
      0005h ???
    CX = printer port (0-2)
    EAX = VM handle
    Return: CF clear if successful
      CF set on error (invalid port number or ???)
      0006h-000Eh unused
    Return: CF set
      000Fh ???
    CX = printer port (0-2)
    AX = ???
    Return: CF clear if successful
      CF set on error (e.g. invalid port number)
      0010h ???
    CX = printer port (0-2)
    Return: CF clear if successful
```

```
      CF set on error (e.g. invalid port number)
      0011h ???
    CX = printer port (0-2)
    Return: CF clear if successful
      CF set on error (e.g. invalid port number)
      0012h get port status
    CX = printer port (0-2)
    Return: CF clear if successful
          AX = port status (see #P0658 at PORT 03BCh"LPT")
      CF set on error (e.g. invalid port number)
      else
    Return: CF set
Note: these functions are apparently only available from the system VM,
    returning CF set
SeeAlso: #02652,#02654
--------W-2F1684BX0010-----------------------
INT 2F - MS Windows - IOS - GET API ENTRY POINT
  AX = 1684h
  BX = 0010h (virtual device ID for IOS device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02654)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02654)
Call IOS entry point with:
  EAX = function number
      0000h ???
    Return: CF clear if successful
          AX = 0000h
      CF set on error
          AX = FFFFh
      0001h check if ???
    Return: CF clear if successful
          AX = 0000h
      CF set on error
          AX = FFFFh
      0002h requestor services???
    DL = service number???
    Return: CF clear if successful
          AX = 0000h
```

```
          DX = ???
      CF set on error
          AX = FFFFh
    Note: calls "IOS_Requestor_Service" (see INT 20"Windows")
      0003h ??? (copies five bytes of data internally)
    Return: CF clear if successful
          AX = 0000h
          EDX = ???
      CF set on error
          AX = FFFFh
      else
    Return: CF set
      AX = FFFFh
SeeAlso: #02653,#02655
--------W-2F1684BX0011-----------------------
INT 2F - MS Windows - VMCPD - GET API ENTRY POINT
  AX = 1684h
  BX = 0011h (virtual device ID for VMCPD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02655)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02655)
Call Windows95 VMCPD protected-mode entry point with:
  EAX = function number
      0000h get VMCPD version
    Return: CF clear
      AH = major version
      AL = minor version
      0001h get ??? flags
    Return: CF clear
      AX = ??? flags
          bit 0: ???
          bit 1: ???
          bit 2: ???
          bit 3: ???
      else
    Return: CF set
SeeAlso: #02654,#02656
--------W-2F1684BX0015-----------------------
```

```
INT 2F - MS Windows - DOSMGR - GET API ENTRY POINT
  AX = 1684h
  BX = 0015h (virtual device ID for DOSMGR device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02656,#01282)
      0000h:0000h if the VxD does not support an API
SeeAlso: #01282 at INT 20"Windows"

(Table 02656)
Call DOSMGR entry point with:
  AX = 0000h get DOSMGR version
      Return: CF clear
        AX = version (AH = major, AL = minor)
  AX = 0001h set critical focus
      Return: CF clear
  AX = 0002h crash current virtual machine
      Return: never
      Note:   displays message box stating that "application has been
          stopped by the DOSMGR device"
  AX = 0003h enter critical section
      Note:   this function assumes that the code for INT 2A/AX=8001h
          and INT 2A/AX=8002h have been modified for Windows
  AX = 0004h get VM ID byte
      Return: CF clear if successful
      ES:DI -> VM ID byte
        CF set on error
      Note:   this function fails if the INT 2A modifications have not
          yet been applied
  AX = 0005h inform Windows of possible media change
      BL = drive number (00h=A:)
      Return: CF clear if successful
        CF set on error
SeeAlso: #01282 at INT 20"Windows",#02655,#02657
--------W-2F1684BX0017-----------------------
INT 2F U - MS Windows - SHELL - GET API ENTRY POINT
  AX = 1684h
  BX = 0017h (virtual device ID for SHELL device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02657)
      0000h:0000h if the VxD does not support an API
SeeAlso: AX=1684h/BX=0021h,#01283 at INT 20"Windows"
```

```
(Table 02657)
Call SHELL entry point with:
  EDX = function number (0000h-0027h,0100h-0108h, mostly unknown)
      0000h get version number
    Return: AX = version number
      EBX = system VM handle
      0001h "SHELL_Get_SYSVM_Info" get system VM information
    Return: CF clear
      AX bit 0 set if system VM executing exclusively
      BX = background time slice priority
      CX = foreground time slice priority
      SI = minimum time slice in milliseconds
      0002h "SHELL_Set_SYSVM_Info" set system VM information
    AX bit 0 set if system VM should execute exclusively (ignored?)
    BX = background time slice priority (1-10000)
    CX = foreground time slice priority (1-10000)
    SI = minimum time slice in milliseconds (1-10000)
    Return: CF clear if successful
      0003h "SHELL_Crt_VM" create a virtual machine
    ES:EDI -> SEB structure (see #02658)
    Return: CF clear if successful
        EAX = VM handle
      CF set on error
        EDX,EAX = result from GetSetDetailedVMError()
      0004h "SHELL_Destroy_VM" destroy a virtual machine
    EBX = VM handle (not system VM)
    Return: nothing
      0005h "SHELL_Set_Focus"
    EBX = VM handle
    ECX = ???
    Return: nothing
      0006h "SHELL_Get_VM_State"
    EBX = VM handle (not system VM)
    ES:EDI -> ??? structure
    Return: CF clear if successful
      0007h "SHELL_Set_VM_State"
    EBX = VM handle (not system VM)
    ES:EDI -> ??? structure
      0008h "SHELL_Debug_Out"
    ???
```

```
Return: ???
Note: dummy function in retail version of MS Windows
  0009h "SHELL_VMDA_Init"
???
Return: ???
  000Ah "SHELL_VMDA_Exit"
???
Return: ???
  000Bh "SHELL_Get_Message_Txt"
???
Return: ???
  000Ch "SHELL_Event_Complete"
???
Return: ???
  000Dh "SHELL_Get_Contention_Info"
???
Return: ???
  000Eh "SHELL_Get_Clip_Info"
???
Return: ???
  000Fh "SHELL_Set_Paste"
???
Return: ???
  0010h "SHELL_Switcher_Assist"
???
Return: ???
  0011h "SHELL_Get_FileSysChng"
???
Return: ???
  0012h "SHELL_Query_Destroy"
???
Return: ???
  0013h "SHELL_SetFocus_Cur_VM" set input focus to current VM
???
Return: ???
  0014h "SHELL_User_Busy_API"
???
Return: ???
  0015h "SHELL_Chng_Hot_Key"
???
Return: ???
```

```
  0016h "SHELL_Get_TermInfo"
???
Return: ???
  ---Windows95---
  0017h ???
  0018h ???
  0019h ???
  001Ah ???
  001Bh ???
  001Ch ???
  001Dh ???
  001Eh ???
  001Fh ???
  0020h ???
  0021h ???
  0022h ???
  0023h ???
  0024h ???
  0025h ???
  0026h ???
Note: makes VxDCALL 00178002h (see INT 20"Windows")
  0027h ???
  0100h get ??? version
Return: AX = version??? (0400h for Windows95)
  0101h not implemented
Return: CF set
  EAX = FFFFFFFFh
  0102h not implemented
Return: CF set
  EAX = FFFFFFFFh
  0103h not implemented
Return: CF set
  EAX = FFFFFFFFh
  0104h ???
  0105h ???
  0106h ???
???
Return: CF clear if successful
  CF set on error
  0107h get SDK version for VxD
AX = VxD identifier
```

```
    Return: EAX = VxD ID (high word) and SDK version (low)
          00000000h if no such VxD loaded
    Note: makes a VMMCALL 0001013Fh (see INT 20"Windows")
        followed by ???
      0108h ???
Return: CF set if called from VM other than system VM
      EAX = FFFFFFFFh
Note: except for functions 0013h,0026h,and 010xh, this API may only be
    called from the system VM
SeeAlso: #01283 at INT 20"Windows"

Format of Shell Execution Block (SEB):
Offset  Size  Description (Table 02658)
 00h  DWORD PIF flags (see #02659)
 04h  DWORD display flags (see #02660)
 08h  PWORD -> pathname of .EXE to run
 0Eh  PWORD -> argument list
 14h  PWORD -> working drive/directory
 1Ah  WORD  desired number of V86 pages for virtual machine
 1Ch  WORD  minimum number of V86 pages for VM
 1Eh  WORD  foreground priority
 20h  WORD  background priority
 22h  WORD  maximum KB of EMS
 24h  WORD  minimum KB of EMS
 26h  WORD  maximum KB of XMS
 28h  WORD  minimum KB of XMS
 2Ah  WORD  maximum KB of DPMI???
 2Ch  WORD  minimum KB of DPMI???
 2Eh 128 BYTEs  title
Note: the PWORDs at offsets 08h,0Eh, and 14h consist of a DWORD offset
    followed by a WORD selector

Bitfields for 386 Enhanced Mode PIF flags:
Bit(s)  Description (Table 02659)
 0  exclusive use of processor when VM is fullscreen
 1  VM runs in background
 2  VM runs in window
 3-4  ???
 5  Alt-Tab reserved
 6  Alt-Esc reserved
 7  Alt-Space reserved
```

```
 8  Alt-Enter reserved
 9  Alt-PrtSc reserved
 10 PrtSc reserved
 11 Ctrl-Esc reserved
 12 VM will release idle time slice
 13 VM not allowed to use high memory
 14 ???
 15 VM expanded memory not pageable
 16 VM extended memory not pageable
 17 Fast paste from clipboard enabled
 18 VM application memory not pageable
 30 Close VM when application exits
SeeAlso: #02658,#02660

Bitfields for SHELL display options:
Bit(s)  Description (Table 02660)
 0  emulate text mode
 1  monitor text port
 2  monitor low graphics port
 3  monitor high graphics port
 7  Retain video memory
SeeAlso: #02658,#02659
--------W-2F1684BX001A-----------------------
INT 2F - MS Windows - VNETWARE - GET API ENTRY POINT
  AX = 1684h
  BX = 001Ah (virtual device ID for VNETWARE device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX001D-----------------------
INT 2F P - MS Windows - WINDEBUG - GET API ENTRY POINT
  AX = 1684h
  BX = 001Dh (virtual device ID for WINDEBUG device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX0021-----------------------
INT 2F PU - MS Windows - PAGEFILE - GET API ENTRY POINT
  AX = 1684h
```

```
  BX = 0021h (virtual device ID for PAGEFILE device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02661)
      0000h:0000h if the VxD does not support an API
SeeAlso: AX=1684h/BX=0017h,#01289 at INT 20"Windows"

(Table 02661)
Call PAGEFILE entry point with:
  AX = function
      0000h get version
    Return: CF clear
      AX = version (AH = major, AL = minor)
      0001h get swap file info
    DS:SI -> 128-byte buffer for swap file full pathname
    DS:DI -> 128-byte buffer for SPART.PAR full pathname
    Return: CF clear
      AL = pager type (see #02662)
      AH = flags
          bit 7: swap file corrupted
      ECX = maximum size of swap file
      DS:SI buffer filled if paging enabled
      DS:DI buffer filled if permanent swap file
      0002h delete permanent swap file on exit
    Return: CF clear
      0003h get current temporary swap file size
    Return: CF clear
      DX:AX = current swap file size in bytes
        0000h:0000h if permanent swap file
Note: this API is only available in protected mode, and may only be called
    from the system VM
SeeAlso: #01289 at INT 20"Windows",#02663

(Table 02662)
Values for MS Windows PAGEFILE pager type:
 00h  paging disabled
 01h  MSDOS
 02h  BIOS
 03h  32-bit disk access
SeeAlso: #02661
--------W-2F1684BX0022-----------------------
INT 2F P - MS Windows - APIX - GET API ENTRY POINT
```

```
  AX = 1684h
  BX = 0022h (virtual device ID for APIX device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02663)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02663)
Call APIX protected-mode entry point with:
  AH = function number
      00h get APIX version
    Return: CF clear
      AH = major version
      AL = minor version
      01h ???
    Return: CF clear
      AX = number of ???
      02h NOP
    Return: CF clear
      03h ???
    Return: CF clear
      AX = 0000h/FFFFh
      else
    Return: CF clear (bug?)
SeeAlso: #02661,#02666
--------W-2F1684BX0026-----------------------
INT 2F P - MS Windows - VPOWERD - GET API ENTRY POINT
  AX = 1684h
  BX = 0026h (virtual device ID for VPOWERD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02664)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02664)
Call VPOWERD.VXD entry point with:
  AX = function number
      0000h get VPOWERD version
    Return: DX = 0000h
      AX = version (AH = major, AL = minor)
      0001h get APM BIOS version
```

```
    Return: DX:AX = APM BIOS version
      0002h get current power management level
    Return: DX:AX = power management level
      0003h enable/disable power management (see INT 15/AX=5308h)
    ??? = new state of power management
    Return: DX:AX = 0000h:0000h if successful
         else error code (see #02665)
      0004h set power state (see INT 15/AX=5307h)
    Return: DX:AX = 0000h:0000h if successful
         else error code (see #02665)
      0005h set system power status
    Return: DX:AX = 0000h:0000h if successful
         else error code (see #02665)
      0006h restore APM power-on defaults (see INT 15/AX=5309h)
    Return: DX:AX = 0000h:0000h if successful
         else error code (see #02665)
      0007h get power status (see INT 15/AX=530Ah)
    Return: ???
      0008h get APM 1.1 power state (see INT 15/AX=530Ch)
    Return: ???
      0009h invoke OEM APM function
    ??? -> buffer containing parameters for INT 15/AX=5380h
    Return: DX:AX = 0000h:0000h or error code (see #02665)
      buffer updated if successful
      000Ah register power handler
    ???
    Return: DX:AX = 0000h:0000h or error code
      000Bh deregister power handler
    ???
    Return: DX:AX = 0000h:0000h or error code (see #02665)
      000Ch Win32 get system power status
      000Dh Win32 set system power status
      else
    Return: DX = 0000h
      AX = 00FFh
SeeAlso: #02663,#02666

(Table 02665)
Values for VPOWERD.VXD error code:
 000000xxh  APM error code
 000000FFh  function number out of range
```

```
 80000001h  ??? (service 05h)
 80000002h  ??? (service 0Dh)
 80000003h  specified NULL buffer pointer (service 07h,08h,09h)
 80000005h  ??? (service 03h)
 80000006h  ??? (service 04h)
 80000007h  ??? (service 05h)
 80000008h  ??? (service 05h)
 80000009h  out of memory (service 0Ah)
 8000000Ah  ??? (service 0Ah)
 8000000Bh  invalid power handler (service 0Bh)
 8000000Ch  unsupported/disabled??? function
SeeAlso: #02664,#01290
--------W-2F1684BX0027----------------------
INT 2F - MS Windows95 - VXDLDR - GET API ENTRY POINT
  AX = 1684h
  BX = 0027h (virtual device ID for VXDLDR device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02666)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02666)
Call VXDLDR entry point with:
  EAX = function number
      0000h get VXDLDR version
    Return: CF clear
      AX = 0000h (successful)
      DH = major version
      DL = minor version
      0001h load device
    DS(???):DX -> ASCIZ path name of dynamically-loadable VxD
      (driver must reside in current directory or Windows
        system directory???)
    ES:DI = 0000h:0000h
    Return: CF clear if successful
          AX = 0000h
          ES:DI -> VxD API entry point
      CF set on error
          AX = error code (see #02667)
      0002h unload device
    EBX = device ID or FFFFFFFFh (Undefined_Device_ID)
```

```
    ---if EBX=FFFFFFFFh ---
    (DS???):DX -> ASCIZ name of dynamically-loadable device
        (case-sensitive)
    Return: CF clear if successful
          AX = 0000h
      CF set on error
          AX = error code (see #02667)
      else
    Return: CF set
      AX = 000Bh
SeeAlso: #02664,#02668

(Table 02667)
Values for VXDLDR error code:
 0000h  successful
 000Bh  invalid function number
SeeAlso: #02666
--------W-2F1684BX0028-----------------------
INT 2F - MS Windows - NDIS - GET API ENTRY POINT
  AX = 1684h
  BX = 0028h (virtual device ID for NDIS device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02668)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02668)
Call NDIS.VXD entry point with:
  ??? = function number
      0000h set ??? to ???
    ??? = new ???
    Return: DX:AX = 0000h:0001h
      0002h ???
    ???
    Return: DX:AX -> ???
      0003h reset ??? to default
    Return: DX:AX = 0000h:0001h
      else
    Return: DX:AX = 0000h:0000h
SeeAlso: #02666,#02669
--------W-2F1684BX002A-----------------------
```

```
INT 2F P - MS Windows - VWIN32 - GET API ENTRY POINT
  AX = 1684h
  BX = 002Ah (virtual device ID for VWIN32 device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02669)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02669)
Call VWIN32.VXD entry point with:
  AH = function number
       00h get VWIN32 version and ???
    Return: CF clear
      AH = major version
      AL = minor version
      EDX = ???
      01h ???
    EBX = ???
    ECX = ???
    Return: CF clear
      EAX = ???
      02h ???
    Return: CF clear
      AX = ??? or 0000h
      03h address allocation
    DS:??? -> buffer containing/for page data
    ECX = length of buffer
    AL = subfunction
        00h reserve page(s)
        01h commit page(s)
        02h decommit page(s)
        03h free page(s)
    Return: CF clear if successful
      CF set on error
    Note: this function uses ECX bytes of stack
      04h get ???
    Return: CF clear
      EAX = ???
      05h ???
    EBX = ???
    Return: CF clear
```

```
  EAX = ???
  06h ???
EBX = ???
Return: CF clear
  EAX = ???
  07h ???
EBX = ???
Return: CF clear
  EAX = ???
  08h get ???
Return: CF clear
  AX = ???
  09h ???
EBX = ???
ECX = ???
Return: CF clear
  0Ah ???
EBX = ???
Return: CF clear
  0Bh ???
EBX = ???
Return: CF clear
  0Ch ???
EBX = ???
ECX = ???
EDX = ???
???
Return: CF clear if successful
      EAX = ???
  CF set on error
  0Dh clear ???
Return: CF clear
  0Eh ???
EBX = ???
ECX = ???
Return: CF clear
  0Fh ???
EBX = ???
ECX = ???
Return: CF clear
  10h ???
```

```
    Return: CF clear
    Note: invokes VMMcall 00010184h
      11h ???
    Return: CF clear
    Note: invokes VMMcall 00010160h
      12h ???
    ???
      13h pop up system error dialogue
    Return: CF clear
      AX = ??? or 0000h
      14h "IFSMgr_GetConversionTablePtrs"
    Return: CF clear
      DX:AX -> ???
    Note: invokes VxDcall 00400051h
      15h "Boost_With_Decay"
    EBX = ???
    ECX = ???
    EDX = ???
    Return: CF clear
      else
    Return: CF set
SeeAlso: #02668,#02670
--------W-2F1684BX002B-----------------------
INT 2F - MS Windows - VCOMM - GET API ENTRY POINT
  AX = 1684h
  BX = 002Bh (virtual device ID for VCOMM device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02670)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02670)
Call VCOMM.VXD entry point with:
  AX = function number
      0000h open COM/LPT port
    BX = port number (00h-7Fh = COMx, 80h-FFh = LPTx)
    Return: DX:AX = handle???
      0001h set comm state
    ???
    Return: AX = ???
      0002h setup comm port
```

```
???
Return: AX = status (0000h failed, FFFFh success)
  0003h transmit character
EBX = handle???
CL = character to transmit
Return: AX = status???
  0004h close comm port
EBX = handle???
Return: ???
  0005h clear comm error
EBX = handle???
EAX = ???
Return: AX = status???
  0006h "EscapeCommFunction"
EBX = handle???
CX = ???
EAX = ???
Return: DX:AX = ???
  0007h purge buffers
EBX = handle???
CX = ???
Return: AX = status???
  0008h set comm event mask
EBX = handle???
CX = new event mask
Return: AX = status???
  0009h get comm event mask
EBX = handle???
Return: AX = current event mask
  000Ah ???
EBX = handle???
Return: ???
  000Bh "WriteComm"
EBX = handle???
CX = number of characters to write
ES???:BX -> buffer (if CX > 1)
SI??? low byte contains character if CX=1
Return: AX = status
  EAX high word may be destroyed
  000Ch "ReadComm"
EBX = handle???
```

```
    CX = number of bytes to read
    ES???:DI -> buffer
    Return: AX = status ???
      ZF = ???
      000Dh set ??? callback
    EBX = handle???
    CX = ???
    DX = ???
    Return: AX = status???
      else
    Return: AX = 0000h
SeeAlso: #02669,#02671
--------W-2F1684BX002D-----------------------
INT 2F P - MS Windows - W32S - GET API ENTRY POINT
  AX = 1684h
  BX = 002Dh (virtual device ID for W32S device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX0030-----------------------
INT 2F P - MS Windows - MACH32 - GET API ENTRY POINT
  AX = 1684h
  BX = 0030h (virtual device ID for MACH32 device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX0032-----------------------
INT 2F - MS Windows - SERVER / VSERVER - GET API ENTRY POINT
  AX = 1684h
  BX = 0032h (virtual device ID for SERVER device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02671)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",#01296 at INT 20"Windows"

(Table 02671)
Call Windows95 VSERVER.VXD protected-mode entry point with:
  AX = function number
      0003h NOP
```

```
    Return: AX = 0000h
      0004h NOP
    Return: AX = 0000h
      0007h NOP
    Return: AX = 0000h
      0008h NOP
    Return: nothing
      000Fh ???
    Return: AX = status
          0000h successful
          0842h on error
      0010h ???
    Return: AX = status
          0000h successful
          0842h on error
      else
    Return: AX = 0032h
SeeAlso: #02670,#02672
--------W-2F1684BX0033-----------------------
INT 2F - MS Windows - CONFIGMG - GET API ENTRY POINT
  AX = 1684h
  BX = 0033h (virtual device ID for CONFIGMG device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02672)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02672)
Call CONFIGMG.VXD entry point with:
  AX = function number
      0000h get CONFIGMG version
    Return: CF clear
      AH = major version
      AL = minor version
      ...
      005Ah
      else
    Return: CF set
      AX = 0020h
SeeAlso: #01297 at INT 20"Windows",#02671,#02673
--------x-2F1684BX0034-----------------------
```

```
INT 2F - Intel Plug-and-Play - CONFIGURATION MANAGER - GET ENTRY POINT
  AX = 1684h
  BX = 0034h (ID for Configuration Manager) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> API entry point (see #02673)
      0000h:0000h if Configuration Manager not loaded
Note: this API is often provided by a DOS device driver, in which case it
    is available whether or not MSWindows is running
Index:  installation check;Plug-and-Play Configuration Manager
SeeAlso: AX=1684h/BX=304Ch

(Table 02673)
Call Configuration Manager entry point with:
  AX = function
      0000h "CM_GetVersion" get supported DDI version
    Return: AH = BCD major version
      AL = BCD minor version
      BX = number of devices identified by configuration
    Note: returns AX = 0000h if no config manager installed
      0001h "CM_GetConfig" get device configuration
    BX = device index
    ES:DI -> buffer for configuration information (see #02675)
    Return: AX = status
          0000h successful
        ES:DI buffer filled
          other error code (0001h = index out of range)
      0002h "CM_LockConfig" lock device configuration
    ES:DI -> configuration information (see #02675)
    Return: AX = status
          0000h successful
        ES:DI buffer filled with assigned config
          0001h resources conflict
          0002h invalid request or configuration info
      0003h "CM_UnlockConfig" unlock device configuration
    ES:DI -> configuration information (see #02675)
    Return: AX = status
          0000h successful
        ES:DI buffer filled with assigned config
          0001h invalid request or configuration info
      0004h "CME_QueryResources" get hot-swappable resources
    ES:DI -> configuration information (see #02675)
```

```
    Return: AX = status (see #02674)
      0005h "CME_AllocResources" remove resources from available pool
    ES:DI -> configuration information (see #02675)
    Return: AX = status (see #02674)
      0006h "CME_DeallocResources" return resources to available pool
    ES:DI -> configuration information (see #02675)
    Return: AX = status (see #02674)
SeeAlso: #01298 at INT 20"Windows",#02672,#02676

(Table 02674)
Values for Configuration Manager status:
 00h  successful
 01h  device not found, configuration error
 02h  I/O port unavailable
 04h  IRQ unavailable
 08h  DMA channel unavailable
 10h  memory range unavailable
SeeAlso: #02673

Format of Configuration Information Structure:
Offset  Size  Description (Table 02675)
 00h  DWORD bus ID
 04h  DWORD device ID
 08h  DWORD serial number
 0Ch  DWORD logical ID
 10h  DWORD flags
---ISA bus---
 14h  BYTE  Card Select Number
 15h  BYTE  logical device number
 16h  WORD  Read Data port
------
 18h  WORD  number of memory windows
 1Ah  9 DWORDs  physical base addresses of memory windows
 3Eh  9 DWORDs  length of memory windows
 62h  9 WORDs memory window attributes
 74h  WORD  number of I/O ports
 76h 20 WORDs I/O port base addresses
 B6h 20 WORDs lengths of I/O port ranges
 F6h  WORD  number of IRQs
 F8h  7 BYTEs IRQ registers
 FFh  7 BYTEs IRQ attributes
```

```
106h  WORD  number of DMA channels
108h  7 BYTEs DMA channels used
10Fh  7 WORDs DMA channel attributes
11Dh  3 BYTEs reserved
SeeAlso: #02673
--------W-2F1684BX0036----------------------
INT 2F - MS Windows - VFBACKUP - GET API ENTRY POINT
  AX = 1684h
  BX = 0036h (virtual device ID for VFBACKUP device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02676)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02676)
Call VFBACKUP.VXD entry point with:
  nothing -- this API is a NOP for the default Windows95 VFBACKUP
SeeAlso: #02673,#01126
--------W-2F1684BX0037----------------------
INT 2F - MS Windows - ENABLE.VXD - GET API ENTRY POINT
  AX = 1684h
  BX = 0037h (virtual device ID for ENABLE device) (see #02677)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02676)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02677)
Call Windows95 ENABLE.VXD entry point with:
  AX = function number
      0000h get ENABLE version
    Return: CF clear
      AX = version (AH = major, AL = minor)
      0001h
    EBX = ???
    Return: ???
      0002h get ???
    Return: CF clear
      DX:AX = ???
      0003h get ???
    Return: CF clear
```

```
      DX:AX = ???
      0004h ???
    EBX = ???
    ECX = ???
    EDX = ???
    Return: CF clear if successful
      CF set on error
      else
    Return: CF set
SeeAlso: #02676,#02678
--------W-2F1684BX0038-----------------------
INT 2F - MS Windows - VCOND - GET API ENTRY POINT
  AX = 1684h
  BX = 0038h (virtual device ID for VCOND device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02678)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02678)
Call VCOND.VXD virtual-86 entry point with:
  AX = function number
      0202h
      0203h
      0204h
      0205h
      0206h
      0207h
      0208h
      0209h
      020Ah
      020Bh
      020Dh
      020Eh
      020Fh
      0210h
      0401h
      0402h
      0403h
      0404h
      0405h
```

```
      else
    NOP
SeeAlso: #02679,#02677

(Table 02679)
Call VCOND.VXD protected-mode entry point with:
  AX = function number
      0301h
      0302h
      0303h
      0304h
      0305h
      0306h
      0307h
      0308h
      else
    NOP
SeeAlso: #02678,#02676
--------W-2F1684BX003B-----------------------
INT 2F - MS Windows - DSVXD - GET API ENTRY POINT
  AX = 1684h
  BX = 003Bh (virtual device ID for DSVXD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX003D-----------------------
INT 2F - MS Windows - BIOS VxD - GET API ENTRY POINT
  AX = 1684h
  BX = 003Dh (virtual device ID for BIOS device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02680)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02680)
Call BIOS.VXD entry point with:
  AX = function number
      0000h get BIOS.VXD version
    Return: CF clear
      AH = major version
```

```
      AL = minor version
      0100h ???
    Return: AX = 0000h
    Note: calls CONFIGMG services 804Eh/804Fh
      0200h ???
    Return: CF clear if successful
          AX = ???
      CF set on error
          AX = error code???
    Note: invokes VxDcall 00290002h
      0300h ???
    Return: CF clear if successful
          AX = ???
      CF set on error
          AX = error code???
      else
    Return: CF set
SeeAlso: #02679,#02681
--------W-2F1684BX003E-----------------------
INT 2F - MS Windows - WSOCK - GET API ENTRY POINT
  AX = 1684h
  BX = 003Eh (virtual device ID for WSOCK device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX011F-----------------------
INT 2F P - MS Windows - VFLATD - GET API ENTRY POINT
  AX = 1684h
  BX = 011Fh (virtual device ID for VFLATD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02681)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",AX=1684/BX=045Dh,INT 20"Windows"

(Table 02681)
Call VFLATD.VXD entry point with:
  DL = function number
      00h get VFLATD version and ???
    Return: CF clear
      EAX = version (AH = major, AL = minor)
```

```
      EBX = ???
      ECX = ???
      EDX = ??? or 00000000h
      01h ???
    AX = ???
    CX = ???
    Return: EAX = ???
      EDX = ???
      02h ???
    ???
      03h ???
    EAX = ???
    EBX = ???
    ESI = ???
    CX = ???
    DH = ???
    Return: EAX = ???
      EDX = ???
      CF clear
      04h ???
    DH = ???
    EAX = ???
    ECX = ???
    Return: CF clear
      EAX = ???
      EDX = ???
      05h ???
    ???
    Note: locks some linear memory and calls fn 02h
      06h ???
    ???
    Return: CF clear if successful
      CF set on error
    Note: calls fn 02h and unlocks some linear memory
      else
    Return: CF set
SeeAlso: #02680
--------W-2F1684BX0200----------------------
INT 2F - MS Windows - VIPX - GET API ENTRY POINT
  AX = 1684h
  BX = 0200h (virtual device ID for VIPX device) (see #02642)
```

```
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX0202-----------------------
INT 2F - MS Windows - WINICE - GET API ENTRY POINT
  AX = 1684h
  BX = 0202h (virtual device ID for WINICE device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX0203-----------------------
INT 2F P - MS Windows - VCLIENT - GET API ENTRY POINT
  AX = 1684h
  BX = 0203h (virtual device ID for VCLIENT device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX0205-----------------------
INT 2F - MS Windows - BCW - GET API ENTRY POINT
  AX = 1684h
  BX = 0205h (virtual device ID for BCW device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX0207-----------------------
INT 2F R - MS Windows - DPMS VxD - GET API ENTRY POINT
  AX = 1684h
  BX = 0207h (virtual device ID for DPMS device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX0234-----------------------
INT 2F - MS Windows - VCOMMUTE - GET API ENTRY POINT
  AX = 1684h
  BX = 0234h (virtual device ID for VCOMMUTE device) (see #02642)
  ES:DI = 0000h:0000h
```

```
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX0442----------------------
INT 2F P - MS Windows - VTDAPI - GET API ENTRY POINT
  AX = 1684h
  BX = 0442h (virtual device ID for VTDAPI device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02682)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02682)
Call VTDAPI.VXD entry point with:
  EAX = function number
      0000h
      0001h
      0002h
      0003h
      0004h
      0005h
      0006h
      0007h
      0008h
      0009h
      000Ah
      000Bh
      else
    Return: nothing???
SeeAlso: #02682
--------W-2F1684BX0444----------------------
INT 2F - MS Windows - VADMAD - GET API ENTRY POINT
  AX = 1684h
  BX = 0444h (virtual device ID for VADMAD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02683)
      0000h:0000h if the VxD does not support an API

(Table 02683)
Call VADMAD entry point with:
  DX = operation
```

```
      0000h set VADMAD mode
    AX = desired mode
      0001h set VADMAD channel
    AX = desired channel
Note: after setting mode/channel, start the DMA operation with an OUT to
    I/O port 0Bh (channels 0-3) or D6h (channels 4-7)
SeeAlso: #01268 at INT 20"Windows"
--------W-2F1684BX0445----------------------
INT 2F - MS Windows - VSBD - GET API ENTRY POINT
  AX = 1684h
  BX = 0445h (virtual device ID for VSBD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX0446----------------------
INT 2F - MS Windows - VADLIBD - GET API ENTRY POINT
  AX = 1684h
  BX = 0446h (virtual device ID for VADLIBD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX0449----------------------
INT 2F P - MS Windows - vjoyd - GET API ENTRY POINT
  AX = 1684h
  BX = 0449h (virtual device ID for "vjoyd" device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02684)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02684)
Call VJOYD.VXD entry point with:
  AX = function number
      0000h get VJOYD version
    Return: AH = major version
      AL = minor version
      0001h ???
    DX = ???
    Return: DX:AX = ???
```

```
      0002h ???
    DX = ???
    Return: DX:AX = ???
      0003h ???
    Retrun: AX = 0001h
      0004h ???
    DX = ???
    Return: DX:AX = ???
      0005h ???
    Return: ???
      else
    Return: EAX = 00000000h
SeeAlso: #02682,#02685
--------W-2F1684BX044A-----------------------
INT 2F - MS Windows - mmdevldr - GET API ENTRY POINT
  AX = 1684h
  BX = 044Ah (virtual device ID for "mmdevldr" device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02685)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02685)
Call MMDEVLDR.VXD entry point with:
  DX = function number
      0000h ???
    Return: CF clear if successful
          AX = 0000h
      CF set on error
          AX = error code (000Bh)
    Note: invokes VxDCall 17000Eh ("CallAtAppyTime")
      0001h ???
    Return: CF clear if successful
          AX = 0000h
      CF set on error
          AX = error code (000Bh)
    Note: invokes VxDCall 17000Eh ("CallAtAppyTime")
      0002h ???
    EDX = ???
    Return: CF clear if successful
          AX = 0000h
```

```
          EDX = ???
      CF set on error
          AX = error code
      0003h ???
    Return: CF clear if successful
          AX = 0000h
      CF set on error
          AX = error code
    Note: invokes VxDcall 2A0002h ("VWIN32_QueueUserApc")
      0004h set Win32 event
    Return: CF clear if successful
          AX = 0000h
      CF set on error
          AX = error code
    Note: invokes VxDcall 2A000Eh ("VWIN32_SetWin32Event")
      0005h ??? (allocates some memory)
    Return: CF clear
      AX = 0000h
      0006h ??? (frees memory)
    Return: CF clear if successful
          AX = 0000h
      CF set on error
          AX = error code
      else
    Return: CF set
      AX = 000Bh (invalid function)
SeeAlso: #02684,#02686
--------W-2F1684BX045D-----------------------
INT 2F P - MS Windows - VflatD - GET API ENTRY POINT
  AX = 1684h
  BX = 045Dh (virtual device ID for VflatD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",AX=1684h/BX=011Fh,INT 20"Windows"
--------W-2F1684BX045F-----------------------
INT 2F - MS Windows - azt16 - GET API ENTRY POINT
  AX = 1684h
  BX = 045Fh (virtual device ID for "azt16" device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02686)
```

```
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h/BX=3110h,AX=1684h"DEVICE API",INT 20"Windows"

(Table 02686)
Call azt16.VXD entry point with:
  DX = function number
      0000h get azt16 version
    Return: CF clear
      AX = version (AH=major, AL=minor)
      0001h ???
    AX = subfunction
        0000h ???
      Return:
        0001h ???
      ECX = ???
        else error
    Return: CF clear if successful
          ???
      CF set on error
          AX = error code
      0002h ???
    AX = ???
    BX = ???
    Return: ???
      0003h ???
    AX = ???
    BX = ???
    Return: ???
      0004h ???
    BX = ???
    CX = ???
    Return: CF clear if successful
           AX = 0001h
      CF set on error
           AX = 0000h
      0005h ???
    BX = ???
    CX = ???
    Return: CF clear if successful
           AX = 0001h
      CF set on error
```

```
        AX = 0000h
  0006h ???
BX = ???
ECX = ???
Return: CF clear if succesful
      AX = ???
  CF set on error
      AX = FFFFh
  0100h get azt16 version
Return: CF clear
  AX = version (AH=major, AL=minor)
  0101h
AX = ???
ECX = ???
Return: CF clear if successful
      AX = 0001h
  CF set on error
      AX = 0000h
  0102h ???
AX = ???
Return: CF clear if successful
  CF set on error
      AX = reason??? (0/1/2)
  0103h ???
AX = ???
Return: CF clear if successful
      AX = 0000h
  CF set on error
      AX = reason??? (1/3)
  0200h ???
EDX = ???
???
Return: CF clear if successful
      DX:AX = ???
  CF set on error
      DX:AX = 0000h:0000h
  0201h ???
???
Return: CF clear
  AX= 0000h
  else
```

```
    Return: CF set
SeeAlso: #02685,#02705
--------W-2F1684BX0460-----------------------
INT 2F P - MS Windows - UNIMODEM - GET API ENTRY POINT
  AX = 1684h
  BX = 0460h (virtual device ID for UNIMODEM device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02687)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02687)
Call UNIMODEM.VXD protected-mode entry point with:
  AX = function number
      0000h
    Return: AX = ???
      0001h
    Return: AX = ???
      0002h
    Return: AX = ???
      0003h
    Return: AX = ???
      0004h
    Return: AX = ???
      0005h
    Return: AX = ???
      0006h
    Return: AX = ???
      0007h
    Return: AX = ???
      else
    Return: AX = 0002h
SeeAlso: #02686,#02688
--------W-2F1684BX0480-----------------------
INT 2F - MS Windows - VNetSup - GET API ENTRY POINT
  AX = 1684h
  BX = 0480h (virtual device ID for VNetSup device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02688)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
```

```
(Table 02688)
Call VNetSup.VXD entry point with:
  AX = function number
       0000h
    Return: AX = ???
       0001h
    Return: AX = ???
       0002h
    Return: AX = ???
       else
    Return: CF set
       AX = 0001h
SeeAlso: #02687,#02689
--------W-2F1684BX0482-----------------------
INT 2F - MS Windows - VBrowse - GET API ENTRY POINT
  AX = 1684h
  BX = 0482h (virtual device ID for VBrowse device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX0483-----------------------
INT 2F - MS Windows - VSHARE - GET API ENTRY POINT
  AX = 1684h
  BX = 0483h (virtual device ID for VSHARE device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02689)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02689)
Call Windows95 VSHARE.VXD entry point with:
  AX = function number
       0000h get VSHARE version
    Return: AH = major version
       AL = (BCD?) minor version
       else
    NOP
SeeAlso: #02688
--------W-2F1684BX0484-----------------------
```

```
INT 2F P - MS Windows - IFSMgr - GET API ENTRY POINT
  AX = 1684h
  BX = 0484h (virtual device ID for IFSMgr device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX0486----------------------
INT 2F - MS Windows - VFAT - GET API ENTRY POINT
  AX = 1684h
  BX = 0486h (virtual device ID for VFAT device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX0487----------------------
INT 2F - MS Windows - NWLINK - GET API ENTRY POINT
  AX = 1684h
  BX = 0487h (virtual device ID for NWLINK device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX0489----------------------
INT 2F R - MS Windows - VIP - GET API ENTRY POINT
  AX = 1684h
  BX = 0489h (virtual device ID for VIP device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX048A----------------------
INT 2F - MS Windows 3.11 - VXDLDR - GET API ENTRY POINT
  AX = 1684h
  BX = 048Ah (virtual device ID for VTCP device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX048A----------------------
INT 2F - MS Windows - VCACHE - GET API ENTRY POINT
```

```
  AX = 1684h
  BX = 048Ah (virtual device ID for VCACHE device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02691)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02690)
Call Windows95 VCACHE.VXD entry point with:
  Return: CF set
SeeAlso: #02689,#02691
--------W-2F1684BX048D-----------------------
INT 2F - MS Windows - RASMAC - GET API ENTRY POINT
  AX = 1684h
  BX = 048Dh (virtual device ID for RASMAC device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX048E-----------------------
INT 2F - MS Windows - NWREDIR - GET API ENTRY POINT
  AX = 1684h
  BX = 048Eh (virtual device ID for NWREDIR device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02691)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02691)
Call Windows95 NWREDIR.VXD entry point with:
  Return: CF set
    EAX = FFFFFFFFh
SeeAlso: #02690
--------W-2F1684BX0494-----------------------
INT 2F - MS Windows - NSCL - GET API ENTRY POINT
  AX = 1684h
  BX = 0494h (virtual device ID for NSCL device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02692,#02693)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
```

```
(Table 02692)
Call Windows95 NSCL.VXD virtual-86 entry point with:
  AL = function number
       00h
       01h
       02h
       03h
       04h
       05h
       06h
       07h
       08h
       09h
       0Ah
       else
    Return: AX = FFFFh
SeeAlso: #02691,#02692

(Table 02693)
Call Windows95 NSCL.VXD protected-mode entry point with:
  AL = function number
       00h
       01h
       02h
       03h
       else
    Return: AX = FFFFh
SeeAlso: #02692
--------W-2F1684BX0499-----------------------
INT 2F - MS Windows - PPPMAC - GET API ENTRY POINT
  AX = 1684h
  BX = 0499h (virtual device ID for PPPMAC device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX049A-----------------------
INT 2F - MS Windows - VDHCP - GET API ENTRY POINT
  AX = 1684h
  BX = 049Ah (virtual device ID for VDHCP device) (see #02642)
```

```
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX049B-----------------------
INT 2F - MS Windows - VNBT - GET API ENTRY POINT
  AX = 1684h
  BX = 049Bh (virtual device ID for VNBT device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX1021----------------------
INT 2F - MS Windows - VMB - GET API ENTRY POINT
  AX = 1684h
  BX = 1021h (virtual device ID for VMB device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX28A0----------------------
INT 2F - MS Windows - PHARLAPX - GET API ENTRY POINT
  AX = 1684h
  BX = 28A0h (virtual device ID for PHARLAPX device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02694)
      0000h:0000h if the VxD does not support an API

(Table 02694)
Call PHARLAPX VxD entry point with:
  AX = function
      0001h get PHARLAP.386 version
    Return: AX = version number (AH = major, AL = minor)
  ---queue functions---
      0101h allocate a new message queue
    CX = size of queue data buffer in bytes
    Return: DX:AX = handle for new queue, or 0000h:0000h on error
      0102h allocate a new key queue
    CX = size of queue data buffer in bytes
    EDX = VM handle into which keys will be pasted
    Return: DX:AX = handle for new queue, or 0000h:0000h on error
```

```
  0103h free message queue
EDX = queue handle
Return: AX = status (0000h,0003h,0007h) (see #02695)
  0104h free key queue
EDX = queue handle
Return: AX = status (0000h,0003h,0005h) (see #02695)
  0105h add message to communications queue
EDX = queue handle
BX = length of message data in bytes
CX = length of message header in bytes
ES:(E)SI -> message header
GS:(E)DI -> message data
Return: AX = status (0000h-0003h,0007h) (see #02695)
  0106h remove message from queue
EDX = queue handle
CX = length of buffer in bytes
ES:(E)SI -> buffer for message
Return: AX = status (0000h,0003h,0006h,0007h,0008h) (see #02695)
  CX = length of returned message (if AX=0000h or 0008h)
  0107h flush queue (remove all data)
EDX = queue handle
Return: AX = status (0000h,0003h) (see #02695)
  0108h add PasteKey structure(s) to key queue
EDX = queue handle
CX = number of PasteKey structures in buffer
ES:(E)SI -> PasteKey array (see #02696)
Return: AX = status (0000h-0003h) (see #02695)
  0109h register enqueueing callback function
EDX = queue handle
ECX = function argument
ES:(E)SI -> callback function
Return: AX = status (0000h,0003h,0009h) (see #02695)
  010Ah register dequeueing callback function
EDX = queue handle
ECX = function argument
ES:(E)SI -> callback function
Return: AX = status (0000h,0003h,0009h) (see #02695)
  010Bh unregister enqueueing callback function
EDX = queue handle
Return: AX = status (0000h,0003h,0009h) (see #02695)
  010Ch unregister dequeueing callback function
```

```
    EDX = queue handle
    Return: AX = status (0000h,0003h,0009h) (see #02695)
      010Dh get message queue status
    EDX = queue handle
    Return: AX = status (0000h,0003h) (see #02695)
      CX = number of pending messages
      010Eh peek at message in queue
    EDX = queue handle
    BX = number of message in queue (0000h = first)
    CX = size of buffer in bytes
    ES:(E)SI -> buffer for message
    Return: AX = status (0000h,0003h,0006h,0008h) (see #02695)
      CX = length of returned message (if AX=0000h or 0008h)
      010Fh peek at last message in queue
    EDX = queue handle
    CX = size of buffer in bytes
    ES:(E)SI -> buffer for message
    Return: AX = status (0000h,0003h,0006h,0008h) (see #02695)
      CX = length of returned message (if AX=0000h or 0008h)
      0110h replace last message in queue
    EDX = queue handle
    CX = length of message header in bytes
    BX = length of message data in bytes
    ES:(E)SI -> message header
    GS:(E)DI -> message data
    Return: AX = status (0000h,0002h,0003h) (see #02695)
      0111h set permitted message count for queue
    EDX = queue handle
    CX = maximum number of messages to enqueue (FFFFh = unlimited)
    Return: AX = status (0000h,0003h) (see #02695)
  ---generalized VxD services---
      0202h call VxD function
    ES:(E)BX -> in/out register-set buffer
    Return: buffer updated
      0203h map flat
    ???
  --system register functions---
      0301h read system registers into buffer
    ES:(E)SI -> 512-byte buffer
    Return: AX = 0000h
      buffer filled (mostly zeros)
```

```
      0302h copy linear memory into buffer
    EDX = linear address
    CX = number of bytes to copy
    ES:(E)SI -> buffer
    Return: AX = 0000h
      0303h copy data into linear memory
    EDX = linear address
    CX = number of bytes to copy
    ES:(E)SI -> buffer
    Return: AX = 0000h
      0304h freeze VM
    ???
      0305h unfreeze VM
    ???
  ---name registration functions---
      0401h register name
    EDX = magic number to associate with name
    ES:(E)SI -> name to register
    Return: AX = status (0000h,0009h) (see #02695)
      0402h unregister name
    ES:(E)SI -> name to be unregistered
    Return: AX = status (0000h,0009h) (see #02695)
      0403h look up name
    ES:(E)SI -> name to look up
    Return: DX:AX = magic number or 0000h:0000h if not registered
      0404h get name list handle
    Return: DX:AX = name list handle
        0000h:0000h if not initialized
  ---special DOS server routines (undocumented)---
      0501h register
      0502h unregister
      0503h validate VM
      0504h get INT9 count
      0505h get screen line
      0506h get shift status
      0507h get server PB pointer
      0508h initialize DOS shell
      0509h get last VM handle

(Table 02695)
Values for PHARLAPX function status:
```

```
 00h  successful
 01h  data is too large to fit in queue
 02h  queue is full
 03h  invalid queue handle
 04h  invalid VM handle for queue
 05h  error starting a paste operation
 06h  queue is empty
 07h  a VM is blocked waiting on the queue
 08h  message was too long (truncated)
 09h  unable to register or unregister specified callback
SeeAlso: #02694

Format of PHARLAPX PasteKey structure:
Offset  Size  Description (Table 02696)
 00h  BYTE  ASCII code
 01h  BYTE  scan code (see #00006)
 02h  WORD  shift states
SeeAlso: #02694

Format of PHARLAPX VxD-call register structure:
Offset  Size  Description (Table 02697)
 00h  DWORD call number
 04h  WORD  input register map (see #02698)
 06h  WORD  output register map (see #02698)
 08h  7 DWORDs  values for EAX, EBX, ECX, EDX, EBP, ESI, EDI on call
 24h  4 WORDs values for DS, ES, FG, GS on call
 2Ch  DWORD EFLAGS on call
 30h  7 DWORDs  returned values of EAX, EBX, ECX, EDX, EBP, ESI, EDI
 4Ch  4 WORDs returned values of DS, ES, FS, GS
 54h  DWORD returned EFLAGS
SeeAlso: #02694

Bitfields for PHARLAPX VxD-call register map:
Bit(s)  Description (Table 02698)
 0  value in EAX field is valid
 1  value in EBX field is valid
 2  value in ECX field is valid
 3  value in EDX field is valid
 4  value in EBP field is valid
 5  value in ESI field is valid
 6  value in EDI field is valid
```

```
 7  value in DS field is valid
 8  value in ES field is valid
 9  value in FS field is valid
 10 value in GS field is valid
 11 value in EFLAGS field is valid
SeeAlso: #02697
--------W-2F1684BX28A1-----------------------
INT 2F - MS Windows - PharLap VxD - GET API ENTRY POINT
  AX = 1684h
  BX = 28A1h (virtual device ID for PharLap device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",AX=1684h/BX=28A0h,INT 20"Windows"
--------W-2F1684BX2925----------------------
INT 2F - MS Windows - EDOS - GET API ENTRY POINT
  AX = 1684h
  BX = 2925h (virtual device ID for EDOS device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02699)
      0000h:0000h if the VxD does not support an API

(Table 02699)
Call EDOS entry point with:
  AX = 0000h get EDOS version number
      Return: AH = major version
        AL = minor version
  AX = 0001h display message
      CX = 0
      DX:BX -> ASCIZ Message
  AX = 0002h get EDOS error coded
      Return: EAX = time in milliseconds that Windows has been running
  AX = 0003h execute windows program
      Return: EAX = cumulative amount of time the virtual machine has
      been active, in milliseconds
  AX = 0008h get/set priority
      BX = 0000h??? foreground
     0001h background
      DI = 0000h get
     0001h set
      DX = priority setting
```

```
      Return: CX = foreground priority
        DX = background priority
        BX:AX = flags
      00000001h exclusive ON
      00000010h background ON
        SI = CPU percentage
--------W-2F1684BX292D-----------------------
INT 2F - MS Windows - VSBPD - GET API ENTRY POINT
  AX = 1684h
  BX = 292Dh (virtual device ID for VSBPD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------W-2F1684BX295A-----------------------
INT 2F - MS Windows - GRVSULTR - GET API ENTRY POINT
  AX = 1684h
  BX = 295Ah (virtual device ID for GRVSULTR device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
--------x-2F1684BX304C-----------------------
INT 2F - Intel Plug-and-Play - CONFIGURATION ACCESS - GET ENTRY POINT
  AX = 1684h
  BX = 304Ch (ID for Configuration Access) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> API entry point (see #02700)
      0000h:0000h if Configuration Access not loaded
Note: this API is often provided by a DOS device driver, in which case it
    is available whether or not MSWindows is running
Index:  installation check;Plug-and-Play Configuration Access
SeeAlso: AX=1684h/BX=0034h

(Table 02700)
Call Plug-and-Play Configuration Access entry point with:
  AX = function
      0000h "CA_GetVersion"
    Return: AX = BCD version (AH = major, AL = minor)
      0001h "CA_PCI_Read_Config_Byte" (see also INT 1A/AX=B108h)
  !!!
```

```
      0002h "CA_PCI_Read_Config_Word" (see also INT 1A/AX=B109h)
      0003h "CA_PCI_Read_Config_DWord" (see also INT 1A/AX=B10Ah)
      0004h "CA_PCI_Write_Config_Byte" (see also INT 1A/AX=B10Bh)
      0005h "CA_PCI_Write_Config_Word" (see also INT 1A/AX=B10Ch)
      0006h "CA_PCI_Write_Config_DWord" (see also INT 1A/AX=B10Dh)
      0007h "CA_PCI_Generate_Special_Cycle" (see also INT 1A/AX=B106h)
      0008h "CA_PCI_Get_Routing_Options" (see also INT 1A/AX=B10Eh)
      0009h invalid function
      000Ah invalid function
      000Bh "CA_PnPISA_Get_Info"
      000Ch "CA_PnPISA_Read_Config_Byte"
      000Dh "CA_PnPISA_Write_Config_Byte"
      000Eh "CA_PnPISA_Get_Resource_Data"
      000Fh invalid function
      0010h "CA_EISA_Get_Board_ID"
      0011h "CA_EISA_Get_Slot_Config"
      0012h "CA_EISA_Get_SlotFunc_Config"
      0013h "CA_EISA_Clear_NVRAM_Config"
      0014h "CA_EISA_Write_Config"
      0015h invalid function
      0016h "CA_ESCD_Get_Info"
      0017h "CA_ESCD_Read_Config"
      0018h "CA_ESCD_Write_Config"
      0019h invalid function
      001Ah "CA_Acfg_PCI_Manage_IRQs"
    DL = IRQ???
    ES:DI -> ???
    Return: AX = status
      001Bh "CA_Acfg_PCI_Get_Routing_Options"
    ES:DI -> IRQ routing table header
        (see #01259 at INT 1A/AX=B406h)
    Return: AX = status
      001Ch-001Fh invalid functions
      0020h "CA_PnPB_Get_Num_Sys_Dev_Nodes"
      0021h "CA_PnPB_Get_Sys_Dev_Node"
      0022h "CA_PnPB_Set_Sys_Dev_Node"
      0023h "CA_PnPB_Get_Stat_Res_Info"
      0024h "CA_PnPB_Set_Stat_Res_Info"
Return: AX = FFFFh if unsupported function
SeeAlso: #02701
--------W-2F1684BX3099----------------------
```

```
INT 2F - MS Windows - VVidramD - GET API ENTRY POINT
  AX = 1684h
  BX = 3099h (virtual device ID for VVidramD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02701)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02701)
Call VVidramD (VIDRAM.VXD) virtual-86 entry point with:
  AX = function number
      0000h map page???
    BX = page number???
    Return: CF clear if successful
      CF set on error
      0001h ???
    Return: CF clear if successful
      CF set on error
      else
    Return: CF set
SeeAlso: #02700,#02702
--------W-2F1684BX30F6-----------------------
INT 2F P - MS Windows - WSVV - GET API ENTRY POINT
  AX = 1684h
  BX = 30F6h (virtual device ID for WSVV device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02702)
Call WSVV.VXD protected-mode entry point with:
  AX = function number
      ????
  Return: ???
SeeAlso: #02701,#02703
--------W-2F1684BX310E----------------------
INT 2F - MS Windows - WPS - GET API ENTRY POINT
  AX = 1684h
  BX = 310Eh (virtual device ID for WPS device) (see #02642)
  ES:DI = 0000h:0000h
```

```
Return: ES:DI -> VxD API entry point (see #02703)
      0000h:0000h if the VxD does not support an API
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02703)
Call WPS protected-mode entry point with:
  DX = function
      0000h get WPS.386 version
    Return: CF clear
      AX = version (AH = major, AL = minor)
      0001h get number of installed VxDs
    Return: CF clear
      AX = number of installed VxDs
      0002h get VxD characteristics
    AX = number of VxD
    ES:BX -> buffer for VxD characteristics structure (see #02704)
    Return: CF clear
      ES:BX buffer filled
SeeAlso: #02702,#02706

Format of WPS.386 VxD characteristics structure:
Offset  Size  Description (Table 02704)
 00h  WORD  VxD ID number
 02h  BYTE  VxD minor version
 03h  BYTE  VxD major version
 04h  BYTE  DDK minor version
 05h  BYTE  DDK major version
 06h  WORD  flags
    bit 0: V86 API supported
    bit 1: PM API supported
    bit 2: services supported
 08h  DWORD start order
 0Ch  9 BYTEs ASCIZ VxD name
SeeAlso: #02703
--------W-2F1684BX3110-----------------------
INT 2F - MS Windows - VSGLX16.386 - GET API ENTRY POINT
  AX = 1684h
  BX = 3110h (virtual device ID for VSGLX16.386) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02705)
      0000h:0000h if the VxD does not support an API
```

```
SeeAlso: AX=1684h/BX=045Fh,AX=1684h"DEVICE API",INT 20"Windows"

(Table 02705)
Call VSGLX16.386 entry point with:
  DX = function number
       0000h get azt16 version
    Return: CF clear
      AX = version returned by "azt16" device
      0001h get ???
    AX = ??? (always fails if nonzero)
    ES:BX -> buffer for ???
      first DWORD of buffer must be set to length of buffer
        (in bytes, 1 <= size <= 92) before calling
    Return: CF clear if successful
          AX = 0001h
      CF set on error (invalid pointer, bad buffer size)
          AX = 0000h
      0002h
    AX = ???
    BX = ???
    Return: CF clear if successful
          AX = ???
      CF set on error
          AX = error code
      0003h
    AX = ???
    BX = ???
    Return: CF clear if successful
      CF set on error
      0004h set ???
    ES:DI -> buffer containing ???
    BX = ???
    CX = number of bytes to copy
    Return: CF clear if successful
          AX = 0001h
      CF set on error
          AX = 0000h
      0005h get ???
    ES:DI -> buffer for ???
    BX = ???
    CX = number of bytes to copy
```

```
    Return: CF clear if successful
          AX = 0001h
      CF set on error
          AX = 0000h
      else
    Return: CF set
SeeAlso: #02686
--------W-2F1684BX31CF----------------------
INT 2F - MS Windows - STAT.386 - GET API ENTRY POINT
  AX = 1684h
  BX = 31CFh (virtual device ID for STAT.386) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02706)
      0000h:0000h if the VxD does not support an API
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02706)
Call STAT.386 entry point with:
  AX = function
      0000h get version
    Return: AX = STAT.386 version (AH = major, AL = minor)
      0001h execute RDMSR/WRMSR/RDTSC
    BH = 00h
    BL = second opcode byte (30h=WRMSR,31h=RDTSC,32h=RDMSR)
    EDX:EDI = value to be written (for BL=30h)
    ECX = MSR number for RDMSR/WRMSR
    Return: EDX:EAX = value read (RDTSR/RDMSR only)
SeeAlso: #02703,#02707
--------W-2F1684BX34DC----------------------
INT 2F - QEMM v8.01 - MAGNARAM VxD - GET API ENTRY POINT
  AX = 1684h
  BX = 34DCh (virtual device ID for MAGNARAM) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02707)
      0000h:0000h if the VxD does not support an API
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02707)
Call MAGNARAM MAGNA95.VXD protected-mode entry point with:
  AX = function number
      0000h get version and ???
```

```
Return: AX = version (AH = major, AL = minor)
  CX = ???
       bit 0: ???
       bit 1: ???
  0001h get ???
Return: CF clear
  DX:AX = ??? SHL 2
  0002h
Return: CF clear if successful
       AX = ???
       DX = ???
  CF set on error
  0003h get ???
Return: CF clear
  DX:AX = ??? SHL 2
  0004h ???
Return: CF clear
  DX:AX = ???
  0005h ???
Return: CF clear
  DX:AX = ???
  0006h ???
Return: CF clear
  DX:AX = ???
  0007h ???
Return: CF clear
  DX:AX = ???
  0008h ???
Return: CF clear
  DX:AX = ???
  0009h ???
Return: CF clear
  DX:AX = ???
  000Ah ???
Return: CF clear
  DX:AX = ???
  000Bh get ???
Return: CF clear
  DX:AX = ??? SHL 2
  000Ch get ???
Return: CF clear
```

```
      DX:AX = ??? SHL 2
      000Dh get ???
    Return: CF clear
      DX:AX = ??? SHL 2
      000Eh get ???
    Return: CF clear
      AX = ???
      DX = ???
      000Fh get ???
    Return: CF clear
      DX:AX = ???
      0010h get ???
    Return: CF clear
      DX:AX = ???
      0011h get ???
    Return: CF clear
      DX:AX = ???
      0012h get ???
    Return: CF clear
      DX:AX = ???
      0013h get ???
    Return: CF clear
      DX:AX = ???
      0014h get ???
    Return: CF clear
      DX:AX = ???
      0015h get ???
    Return: CF clear
      DX:AX = ???
      else
    Return: CF set
SeeAlso: #02706,#02708
--------W-2F1684BX357E-----------------------
INT 2F - MS Windows - DSOUND - GET API ENTRY POINT
  AX = 1684h
  BX = 357Eh (virtual device ID for DSOUND device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"
----------2F1684BX377B-----------------------
```

```
INT 2F - MS Windows - MX1501HAD - GET API ENTRY POINT
  AX = 1684h
  BX = 377Bh (virtual device ID for MX1501HAD device)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02708)
      0000h:0000h if the VxD does not support an API
Note: The drivers VCMD95C.VXD and VCMD.386 are part of the driver disks
    provided with the chip-card-reader/keyboard combination MX 1501 HAD,
    produced by Cherry
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02708)
Call CHERRY VCMD95C.VXD entry point with:
  AX = function
      0001h get version
    Return: AX = version number (0100h) (AH = major, AL = minor)
      0002h hook INT 09 (and 8???)
      0003h unhook INT 09 (and 8???)
      0004h get number of bytes in FIFO
    Return: AX = bytes in FIFO
      0005h get next FIFO-data
    Return: AX = data
      BL = port number
      BH = direction (1=in, 0=out)
      DX:CX = timestamp
      0006h clear FIFO
      0007h output byte
    DX = port number
    BL = keyboard command
    Return: data in FIFO (see #02710)
           (value, port, in/out, timestamp)
      0008h input byte
    DX = port number
    Return: data in FIFO (see #02710)
      (value, port, in/out, timestamp)
      0009h input byte immediately
    DX = port number
    Return: AX = data
      000Ah read next FIFO data (nondestructive)
    Return: AX = data
      BL = port number
```

```
      BH = direction (1=in, 0=out)
      DX:CX = timestamp
      000Bh get timestamp
    Return: DX:CX = timestamp (in ms)
      000Ch enable IRQ 1
      000Dh disable IRQ 1
      000Eh enable data retrieval
    Note: Sets a flag in the internal mode-byte which
        tells the driver to recognize the data
      000Fh disable data retrieval
    Note: resets a flag in the internal mode-byte
      0010h get retrieval mode
    Return: AX = current retrieval mode
      0011h set retrieval mode
    BX = new retrieval mode (see #02709)
    Return: AX = old retrieval mode
      0012h get command value
    Return: AX = command value
      0013h set command value
    BX = command value
SeeAlso: #02706,#02711

Bitfields for retrieval mode:
Bit(s)  Description (Table 02709)
 0  enable data retrieval
 1  0 = interrupt-driven
  1 = polling mode
 2  0 = read port 60h everytime
  1 = read port 60h only when OBF of port 64h is set
 3  0 = don't call old INT 9
  1 = call INT 9 before our INT-handler
 4-7  reserved
SeeAlso: #02708,#02710

Format of FIFO entry (1024 entries in FIFO):
Offset  Size  Description (Table 02710)
 00h  BYTE  data byte
 01h  BYTE  I/O port
 02h  BYTE  direction (1=in, 0=out)
 03h  BYTE  reserved
 04h  DWORD timestamp
```

```
SeeAlso: #02708,#02709
--------W-2F1684BX38DA-----------------------
INT 2F - MS Windows - VIWD - GET API ENTRY POINT
  AX = 1684h
  BX = 38DAh (virtual device ID for VIWD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02711)
      0000h:0000h if the VxD does not support API in current mode
SeeAlso: AX=1684h"DEVICE API",INT 20"Windows"

(Table 02711)
Call VIWD.VXD entry point with:
  DX = function number
      0000h ???
    Return: CF clear
      AX = ???
      0004h ???
    Return: CF clear
      DX = 0000h
      0006h
    Return: CF clear
      000Ah
    AX = ???
    Return: CF clear if successful
      CF set on error
      000Ch
      000Dh
      000Eh
    Return: CF clear
      000Fh
    Return: CF clear
      0010h
      0011h
      0015h
    Return: CF clear if successful
          AX = ???
      CF set on error
          AX = ???
      DX = 0000h
      0016h
      0017h
```

```
    Return: CF clear if successful
          AX = ???
      CF set on error
          AX = ???
      DX = 0000h
      0018h ???
    CX = ???
    Return: CF clear if successful
          AX = 0000h
      CF set on error
      else
    Return: CF set
SeeAlso: #02708,#02712
--------W-2F1684BX4321-----------------------
INT 2F - MS Windows - POSTMSG - GET API ENTRY POINT
  AX = 1684h
  BX = 4321h (virtual device ID for POSTMSG device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02712,#02714)
      0000h:0000h if the VxD does not support an API

(Table 02712)
Call POSTMSG protected-mode entry point with:
  AX = window handle
  CX:BX -> callback procedure (see #02713)
Return: nothing
Note: this call registers a WinApp with the VxD; the callback must be in a
    fixed, non-discardable code segment
SeeAlso: #02714,#02715

(Table 02713)
Values POSTMSG callback routine is called with:
  STACK:  DWORD "lParam" parameter from DOSApp
    WORD  "wParam" parameter from DOSApp
    WORD  Windows message number (WM_USER + 100h)
    WORD  registered HWND

(Table 02714)
Call POSTMSG V86-mode entry point with:
  BX = wParam value to pass to protected-mode callback
  DX:AX = lParam value to pass to protected-mode callback
```

```
Return: CF clear if successful
  CF set on error (no WinApp registered)
SeeAlso: #02712
--------W-2F1684BX7FE0-----------------------
INT 2F - MS Windows - VSWITCHD - GET API ENTRY POINT
  AX = 1684h
  BX = 7FE0h (virtual device ID for VSWITCHD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02715)
      0000h:0000h if the VxD does not support an API

(Table 02715)
Call VSWITCHD entry point with:
  AX = function
      0000h toggle windowed mode (simulate Alt-Enter keypress)
    Return: nothing
      0001h get windowed mode
    Return: CF clear if VM is windowed
      CF set if VM is full-screen
SeeAlso: #02712,#02716
--------W-2F1684BX8888-----------------------
INT 2F - MS Windows - VbillD - GET API ENTRY POINT
  AX = 1684h
  BX = 8888h (virtual device ID for VbillD device) (see #02642)
  ES:DI = 0000h:0000h
Return: ES:DI -> VxD API entry point (see #02716)
      0000h:0000h if the VxD does not support an API

(Table 02716)
Call VbillD entry point with:
  AX = function
      0001h set reverse video
      0002h set normal video
Return: ???
SeeAlso: #02715
--------W-2F1685-----------------------------
INT 2F - MS Windows - SWITCH VMs AND CALLBACK
  AX = 1685h
  BX = VM ID of virtual machine to switch to
  CX = flags (see #02717)
  DX:SI = priority boost (refer to VMM.INC)
```

```
  ES:DI -> FAR procedure to callback
Return: CF set on error
      AX = error code
    01h invalid VM ID
    02h invalid priority boost
    03h invalid flags
  CF clear if successful
      event will be or has been called
Notes:  some DOS devices, such as networks, need to call functions in a
    specific VM. This call forces the appropriate VM to be installed.
  the callback procedure must preserve all registers and return with IRET
SeeAlso: AX=1683h,INT 15/AX=1117h,AX=DB06h"WINGO"

Bitfields for VM switching flags:
Bit(s)  Description (Table 02717)
 0  wait until interrupts enabled
 1  wait until critical section unowned
 2-15 reserved (zero)
--------E-2F1686-----------------------------
INT 2F - DOS Protected-Mode Interface - DETECT MODE
  AX = 1686h
Return: AX = 0000h if operating in protected mode under DPMI (INT 31 available)
  AX nonzero if in real/V86 mode or no DPMI (INT 31 not available)
SeeAlso: AX=1687h
--------E-2F1687-----------------------------
INT 2F - DOS Protected-Mode Interface - INSTALLATION CHECK
  AX = 1687h
Return: AX = 0000h if installed
      BX = flags
    bit 0: 32-bit programs supported
      CL = processor type (02h=80286, 03h=80386, 04h=80486)
      DH = DPMI major version
      DL = two-digit DPMI minor version (binary)
      SI = number of paragraphs of DOS extender private data
      ES:DI -> DPMI mode-switch entry point (see #02718)
  AX nonzero if not installed
SeeAlso: AX=1686h,AX=43E0h,AX=DE01h/BX=4450h,AX=FB42h/BX=0001h
SeeAlso: INT 31/AX=0400h,INT 31/AX=5702h,INT D4/AH=10h

(Table 02718)
Call DPMI mode switch entry point with:
```

```
  AX = flags
       bit 0: set if 32-bit program
  ES = real mode segment of buffer for DPMI private data (ignored if
    SI was zero)
Return: CF set on error
      program still in real mode
      AX = error code (DPMI 1.0+)
         8011h unable to allocate all necessary descriptors
         8021h 32-bit program specified, but 16-bit DPMI host
  CF clear if successful
      CS = 16-bit selector corresponding to real-mode CS
      SS = selector corresponding to real-mode SS (64K limit)
      DS = selector corresponding to real-mode DS (64K limit)
      ES = selector to program's PSP (100h byte limit)
      FS = GS = 0
      high word of ESP = 0 if 32-bit program
      program now in protected mode
Note: this entry point is only called for the initial switch to protected
    mode
--------W-2F1688BX0BAD-----------------------
INT 2F U - MS Windows 3.0, 386MAX v6.01 - GET ALIAS SELECTOR TO LDT
  AX = 1688h
  BX = 0BADh
Return: AX = 0000h if supported
      BX = alias selector for LDT
Note: use the LSL instruction or GetSelectorLimit() to find LDT size
  this call should be considered obsolete for Windows 3.1+, as the
    alias selector can be retrieved via the API entry point for
    "MS-DOS" retrieved from INT 2F/AX=168Ah (see #02720)
SeeAlso: AX=1687h,#02720
--------W-2F1689-----------------------------
INT 2F U - MS Windows 3.0+ - KERNEL IDLE CALL
  AX = 1689h
  BX = status flags (see #04105)
Return: ???
Desc: the Windows KERNEL idle loop calls this function, which VMM uses as an
    indication that the system is idle, which in turn generates INT 28
    and INT 2F/AX=1607h/BX=0018h callouts
SeeAlso: AX=1680h,AX=1607h/BX=0018h,INT 15/AX=1000h,INT 28

Bitfields for Kernel Idle status flags:
```

```
Bit(s)  Description (Table 04106)
 15-1 reserved
 0  "Win_Idle_Mouse_Busy"
--------E-2F168A-----------------------------
INT 2F - DPMI 0.9+ - GET VENDOR-SPECIFIC API ENTRY POINT
  AX = 168Ah
  DS:(E)SI = selector:offset of ASCIZ vendor name (see #02719)
Return: AL = status
      00h successful
         ES:(E)DI -> extended API entry point
      8Ah unsuccessful
Notes:  the vendor name is used to determine which entry point to return; it is
    case-sensitive
  available in protected mode only
  32-bit applications use ESI and EDI, 16-bit applications use SI and DI
  this call is present but not documented for DPMI 0.9
  the Borland C++ 3.1 DPMILOAD does not handle requests for entry points
    other than the MS-DOS one gracefully, producing an unhandled
    exception report; this has been fixed in the Borland Pascal 7 version
SeeAlso: AX=1687h,INT 31/AX=0A00h,INT 31/AH=57h

(Table 02719)
Values for DPMI vendor-specific API names:
 "MS-DOS" MS Windows and 386MAX v6.00+ (see #02720)
 "386MAX" 386MAX v6.00+
 "HELIX_DPMI" Helix Netroom's DPMI server
 "Phar Lap" Phar Lap 286|DOS-Extender RUN286 (see #02721)
 "RATIONAL DOS/4G"  DOS/4G, DOS/4GW
 "VIRTUAL SUPPORT"  Borland 32RTM

(Table 02720)
Call Windows-support ("MS-DOS") entry point with:
  AX = 0100h get LDT alias selector
Return: CF clear if successful
      AX = alias selector
  CF set on error
SeeAlso: #02719,AX=1688h/BX=0BADh

(Table 02721)
Call Phar Lap RUN286 entry point with:
  AX = 0000h (function "load MSW")
```

```
  BX = new value for MSW register (low word of CR0)
Return: ???
SeeAlso: #02719
--------W-2F168B-----------------------------
INT 2F - MS Windows 3.1 - SET FOCUS TO SPECIFIED VIRTUAL MACHINE
  AX = 168Bh
  BX = virtual machine ID (see AX=1683h), 0000h for current DOS box
Return: AL = 00h if focus set to specified VM
Notes:  documented on the Microsoft Developer's Network CD-ROM
  if the VM is a windowed DOS box, it will be set to full screen
SeeAlso: AX=1683h
--------W-2F168C-----------------------------
INT 2F - MS Windows 3.1 - RESTART COMMAND
  AX = 168Ch
  ???
Return: ???
Note: WIN.COM executes specified application
--------W-2F168EDX0000----------------------
INT 2F - Windows95 - TITLE - SET APPLICATION TITLE
  AX = 168Eh
  DX = 0000h
  ES:DI -> ASCIZ application title (max 79 chars+NUL)
Return: AX = status
      0000h failed
      0001h successful
Note: if ES:DI is 0000h:0000h or points at an empty string, the current
    title is removed
BUG:  this function can return a successful status even though the title was
    not changed; reportedly, waiting for two clock ticks after program
    startup solves this problem
SeeAlso: AX=168Eh/DX=0001h,AX=168Eh/DX=0002h
--------W-2F168EDX0001----------------------
INT 2F - Windows95 - TITLE - SET VIRTUAL MACHINE TITLE
  AX = 168Eh
  DX = 0001h
  ES:DI -> ASCIZ virtual machine title (max 29 chars+NUL)
Return: AX = status
      0000h failed
      0001h successful
Notes:  if ES:DI is 0000h:0000h or points at an empty string, the current
    title is removed
```

```
  the VM title should only be changed on explicit instruction from the
    user
BUG:  this function can return a successful status even though the title was
    not changed; reportedly, waiting for two clock ticks after program
    startup solves this problem
SeeAlso: AX=168Eh/DX=0000h,AX=168Eh/DX=0003h
--------W-2F168EDX0002-----------------------
INT 2F - Windows95 - TITLE - GET APPLICATION TITLE
  AX = 168Eh
  DX = 0002h
  ES:DI -> buffer for ASCIZ application title
  CX = size of buffer in bytes
Return: AX = status
      0000h failed
      0001h successful
Desc: copy as much of the application's window title as possible to the given
    buffer, appending a terminating NUL to the buffer
SeeAlso: AX=168Eh/DX=0000h,AX=168Eh/DX=0003h
--------W-2F168EDX0003-----------------------
INT 2F - Windows95 - TITLE - GET VIRTUAL MACHINE TITLE
  AX = 168Eh
  DX = 0003h
  ES:DI -> buffer for ASCIZ virtual-machine title
  CX = size of buffer in bytes
Return: AX = status
      0000h failed
      0001h successful
Desc: copy as much of the virtual machine's title as possible to the given
    buffer, appending a terminating NUL to the buffer
SeeAlso: AX=168Eh/DX=0001h,AX=168Eh/DX=0002h
--------W-2F168FDH00------------------------
INT 2F - Windows95 - CLOSE-AWARENESS - ENABLE/DISABLE CLOSE COMMAND
  AX = 168Fh
  DH = 00h
  DL = new state
      00h disabled
      01h enabled
Return: AX = status
      0000h successful
      else failed
Desc: enable or disable the system menu Close command for an application
```

```
SeeAlso: AX=168Fh/DH=01h,AX=168Fh/DH=02h
--------W-2F168FDH01-------------------------
INT 2F - Windows95 - CLOSE-AWARENESS - QUERY CLOSE
  AX = 168Fh
  DH = 01h
  DL = 00h (reserved)
Return: AX = status
      0000h Close command selected but not yet acknowledged
      0001h Close command issued and acknowledged
      168Fh Close command not selected -- application should continue
Desc: determine whether the user has requested that the application be closed
    by selecting the system menu's Close option
SeeAlso: AX=168Fh/DH=00h,AX=168Fh/DH=02h
--------W-2F168FDH02-------------------------
INT 2F - Windows95 - CLOSE-AWARENESS - ACKNOWLEDGE CLOSE
  AX = 168Fh
  DH = 02h
  DL = 00h (reserved)
Return: AX = status
      0000h successful
      else failed
Note: once a Close command has been issued, no further keyboard input is
    available to the application until it calls this function to
    acknowledge the Close request
SeeAlso: AX=168Fh/DH=00h,AX=168Fh/DH=03h
--------W-2F168FDH03-------------------------
INT 2F - Windows95 - CLOSE-AWARENESS - CANCEL CLOSE
  AX = 168Fh
  DH = 03h
  DL = 00h (reserved)
Return: AX = status
      0000h successful
      else failed
Desc: cancels a close request which has already been acknowledged if the
    application determines that it will not exit at this time
SeeAlso: AX=168Fh/DH=00h,AX=168Fh/DH=03h
--------D-2F1690----------------------------
INT 2F U - MS-DOS 7 kernel - GET/SET ???
  AX = 1690h
  ES:BX -> ???
Return: ES:BX -> ??? data (see #02722)
```

```
SeeAlso: AX=1611h,AX=1614h

Format of MS-DOS 7 kernel ??? data:
Offset  Size  Description (Table 02722)
 00h  DWORD -> ??? data (appears to list the installed drivers)
 04h  DWORD -> ??? (value passed in via ES:BX is stored here)
--------W-2F1700----------------------------
INT 2F - MS Windows "WINOLDAP" - IDENTIFY WinOldAp VERSION
  AX = 1700h
Return: AX = 1700h if this version of WINOLDAP doesn't support clipboard
  AX <> 1700h
      AL = WINOLDAP major version
      AH = WINOLDAP minor version
Program: WinOldAp (WINOLDAP.MOD) is a Microsoft Windows extension supporting
    "old" (character-mode) application access to Dynamic Data Exchange,
    menus, and the Windows clipboard.
Note: this installation check DOES NOT follow the format used by other
    software of returning AL=FFh
SeeAlso: AX=1701h,AX=4601h
Index:  installation check;WINOLDAP
--------W-2F1701----------------------------
INT 2F - MS Windows "WINOLDAP" - OPEN CLIPBOARD
  AX = 1701h
Return: AX = status
      nonzero success
      0000h   clipboard is already open
SeeAlso: AX=1700h,AX=1702h,AX=1703h,AX=1704h,INT 16/AX=CB00h
--------W-2F1702----------------------------
INT 2F - MS Windows "WINOLDAP" - EMPTY CLIPBOARD
  AX = 1702h
Return: AX = status
      nonzero clipboard has been emptied
      0000h   failure
SeeAlso: AX=1700h,AX=1701h,AX=1703h,AX=1704h,INT 16/AX=CB05h
--------W-2F1703----------------------------
INT 2F - MS Windows "WINOLDAP" - SET CLIPBOARD DATA
  AX = 1703h
  DX = clipboard format supported by WinOldAp (see #02723)
  ES:BX -> data (see #02724,#02725)
  SI:CX = size of data
Return: AX = status
```

```
      nonzero data copied into the Clipboard
      0000h   failure
SeeAlso: AX=1701h,AX=1705h,INT 16/AX=CB04h

(Table 02723)
Values for WinOldAp clipboard format:
 01h  text
 02h  bitmap
 03h  metafile picture
 04h  SYLK
 05h  DIF
 06h  TIFF
 07h  OEM text
 08h  DIB bitmap
 80h  special format (used by Windows WRITE, maybe other Windows applets???)
 81h  DSP text
 82h  DSP bitmap

Format of Windows Clipboard bitmap:
Offset  Size  Description (Table 02724)
 00h  WORD  type (0000h)
 02h  WORD  width of bitmap in pixels
 04h  WORD  height of bitmap in pixels
 06h  WORD  bytes per line
 08h  BYTE  number of color planes
 09h  BYTE  number of adjacent color bits in pixel
 0Ah  DWORD pointer to start of data
 0Eh  WORD  width in 0.1mm units
 10h  WORD  height in 0.1mm units
 12h  N BYTEs bitmap data

Format of Windows metafile picture:
Offset  Size  Description (Table 02725)
 00h  WORD  mapping mode
 02h  WORD  X extent
 04h  WORD  Y extent
 06h  WORD  picture data
--------W-2F1704----------------------------
INT 2F - MS Windows "WINOLDAP" - GET CLIPBOARD DATA SIZE
  AX = 1704h
  DX = clipboard format supported by WinOldAp (see #02723)
```

```
Return: DX:AX = size of data in bytes, including any headers
    0000h:0000h if no data in this format in the Clipboard
Note: Windows reportedly rounds up the size of the data to a multiple of 32
    bytes
SeeAlso: AX=1700h,AX=1703h,AX=1705h
--------W-2F1705----------------------------
INT 2F - MS Windows "WINOLDAP" - GET CLIPBOARD DATA
  AX = 1705h
  DX = clipboard format supported by WinOldAp (see #02723)
  ES:BX -> buffer
Return: AX = status
      nonzero success
      0000h   error, or no data in this format in Clipboard
SeeAlso: AX=1700h,AX=1704h,INT 16/AX=CB03h
--------W-2F1708----------------------------
INT 2F - MS Windows "WINOLDAP" - CloseClipboard
  AX = 1708h
Return: AX = status
      0000h failure
      nonzero success
--------W-2F1709----------------------------
INT 2F - MS Windows "WINOLDAP" - COMPACT CLIPBOARD
  AX = 1709h
  SI:CX = desired size in bytes
Return: DX:AX = number of bytes in largest block of free memory
Note: WinOldAp is responsible for including the size of any headers
--------W-2F170A----------------------------
INT 2F - MS Windows "WINOLDAP" - GET DEVICE CAPABILITIES
  AX = 170Ah
  DX = GDI information index (see #02726)
Return: AX = integer value of the desired item
        (see #02727,#02728,#02729,#02730,#02731,#02732,#02733)
Note: This function returns the device-capability bits for the given display

(Table 02726)
Values for GDI information index:
 00h  device driver version
 02h  device classification
 04h  width in mm
 06h  height in mm
 08h  width in pixels
```

```
 0Ah  height in pixels
 0Ch  bits per pixel
 0Eh  number of bit planes
 10h  number of brushes supported by device
 12h  number of pens supported by device
 14h  number of markers supported by device
 16h  number of fonts supported by device
 18h  number of colors
 1Ah  size required for device descriptor
 1Ch  curve capabilities
 1Eh  line capabilities
 20h  polygon capabilities
 22h  text capabilities
 24h  clipping capabilities
 26h  bitblt capabilities
 28h  X aspect
 2Ah  Y aspect
 2Ch  length of hypotenuse of aspect
 58h  logical pixels per inch of width
 5Ah  logical pixels per inch of height
SeeAlso: #02727,#02728,#02729,#02730,#02731,#02732,#02733

(Table 02727)
Values for device classification:
 00h  vector plotter
 01h  raster display
 02h  raster printer
 03h  raster camera
 04h  character-stream, PLP
 05h  Metafile, VDM
 06h  display-file
SeeAlso: #02726,#02728,#02729,#02730,#02731,#02732,#02733

Bitfields for curve capabilities:
Bit(s)  Description (Table 02728)
 0  circles
 1  pie wedges
 2  chord arcs
 3  ellipses
 4  wide lines
 5  styled lines
```

```
 6  wide styled lines
 7  interiors
SeeAlso: #02726,#02727,#02729,#02730,#02731,#02732,#02733

Bitfields for line capabilities:
Bit(s)  Description (Table 02729)
 1  polylines
 2  markers
 3  polymarkers
 4  wide lines
 5  styled lines
 6  wide styled lines
 7  interiors
SeeAlso: #02726,#02727,#02728,#02730,#02731,#02732,#02733

Bitfields for polygon capabilities:
Bit(s)  Description (Table 02730)
 0  polygons
 1  rectangles
 2  trapezoids
 3  scanlines
 4  wide borders
 5  styled borders
 6  wide styled borders
 7  interiors
SeeAlso: #02726,#02727,#02728,#02729,#02731,#02732,#02733

Bitfields for text capabilities:
Bit(s)  Description (Table 02731)
 0  output precision character
 1  output precision stroke
 2  clippping precision stroke
 3  90-degree character rotation
 4  arbitrary character rotation
 5  independent X and Y scaling
 6  double-size
 7  integer scaling
 8  continuous scaling
 9  bold
 10 italic
 11 underline
```

```
 12 strikeout
 13 raster fonts
 14 vector fonts
 15 reserved
SeeAlso: #02726,#02727,#02728,#02729,#02730,#02732,#02733

(Table 02732)
Values for clipping capabilities:
 00h  none
 01h  clipping to rectangles
SeeAlso: #02726,#02727,#02728,#02729,#02730,#02731,#02733

Bitfields for raster capabilities:
Bit(s)  Description (Table 02733)
 0  simple bitBLT
 1  device requires banding support
 2  device requires scaling support
 3  supports >64K bitmap
SeeAlso: #02726,#02727,#02728,#02729,#02730,#02731,#02732
----------2F18------------------------------
INT 2F U - MS-Manager
  AH = 18h
  ???
Return: ???
--------l-2F1900----------------------------
INT 2F U - DOS 4.x only SHELLB.COM - INSTALLATION CHECK
  AX = 1900h
Return: AL = status
      00h not installed
      FFh installed
--------l-2F1901----------------------------
INT 2F U - DOS 4.x only SHELLB.COM - SHELLC.EXE INTERFACE
  AX = 1901h
  BL = SHELLC type
      00h transient
      01h resident
  DS:DX -> far call entry point for resident SHELLC.EXE
Return: ES:DI -> SHELLC.EXE workspace within SHELLB.COM
Note: SHELLB.COM and SHELLC.EXE are parts of the DOS 4.x shell
--------l-2F1902----------------------------
INT 2F U - DOS 4.x only SHELLB.COM - COMMAND.COM INTERFACE
```

```
  AX = 1902h
  ES:DI -> ASCIZ full filename of current batch file, with at least the
       final filename element uppercased
  DS:DX -> buffer for results
Return: AL = 00h  failed, either
     (a) final filename element quoted at ES:DI does not match
           identity of shell batch file quoted as parameter of most
           recent call of SHELLB command, or
     (b) no more Program Start Commands available.
  AL= FFh  success, then:
    memory at DS:[DX+1] onwards filled as:
    DX+1: BYTE  count of bytes of PSC
    DX+2: N BYTEs Program Start Command text
      BYTE  0Dh terminator
Desc: COMMAND.COM executes the result of this call in preference to
     reading a command from a batch file.  Thus the batch file does not
     advance in execution for so long as SHELLB provides PSCs from its
     workspace.
Note: The PSCs are planted in SHELLB workspace by SHELLC, the user
     menu interface.  The final PSC of a sequence is finished with a
     GOTO COMMON, which causes a loop back in the batch file which called
     SHELLC so as to execute SHELLC again.  The check on batch file name
     permits PSCs to CALL nested batch files while PSCs are still stacked
     up for subsequent execution.
--------l-2F1903-----------------------------
INT 2F U - DOS 4.x only SHELLB.COM - COMMAND.COM interface
  AX = 1903h
  ES:DI -> ASCIZ batch file name as for AX=1902h
Return: AL = status
      FFh quoted batch file name matches last SHELLB parameter
      00h it does not
--------l-2F1904-----------------------------
INT 2F U - DOS 4.x only SHELLB.COM - SHELLB transient to TSR interface
  AX = 1904h
Return: ES:DI -> name of current shell batch file:
    WORD  number of bytes of name following
    BYTEs (8 max) uppercase name of shell batch file
----------2F1980-----------------------------
INT 2F U - IBM ROM-DOS v4.0 - INSTALLATION CHECK
  AX = 1980h
Return: AL = FFh if ??? installed/supported
```

```
Note: called at the very beginning of SHELLSTB.COM, which exits if AL is not
    FFh on return
SeeAlso: AX=1981h,AX=1982h
----------2F1981-----------------------------
INT 2F U - IBM ROM-DOS v4.0 - GET ??? STRING
  AX = 1981h
  DS:DX -> buffer for ???
Return: AL = status
      FFh if successful
    DS:DX buffer filled (refer to note below)
      81h on error
Note: the first byte of the buffer is unchanged; depending on a byte in
    IBMBIO.COM, the remainder of the buffer is filled with either
    "C:\ROMSHELL.COM",0Dh or xxh,xxh,0Fh,"C:\ROMSHELL.COM",0Dh
SeeAlso: AX=1980h,AX=1982h
----------2F1982-----------------------------
INT 2F U - IBM ROM-DOS v4.0 - GET ??? TABLE
  AX = 1982h
Return: AL = FFh if supported
      ES:DI -> ??? table (see #02734)
Note: called by ROMSHELL.COM
SeeAlso: AX=1980h,AX=1981h

Format of ROM-DOS v4.0 ??? table:
Offset  Size  Description (Table 02734)
 00h  BYTE  ??? (00h)
 01h  BYTE  ??? (41h) (ROMSHELL.COM checks if =00h)
 02h  BYTE  ??? (00h) (ROMSHELL.COM checks if =01h)
 03h  WORD  ??? (0001h) (ROMSHELL.COM checks if =0001h)
 05h  BYTE  ??? (00h)
 06h  WORD  ??? (04D5h)
--------V-2F1A00-----------------------------
INT 2F - DOS 4.0+ ANSI.SYS - INSTALLATION CHECK
  AX = 1A00h
Return: AL = FFh if installed
Notes:  AVATAR.SYS also responds to this call
  documented for DOS 5+, but undocumented for DOS 4.x
--------V-2F1A00BX414E-----------------------
INT 2F - ANSIPLUS.SYS v2.00+ - INSTALLATION CHECK
  AX = 1A00h
  BX = 414Eh ('AN')
```

```
  CX = 5349h ('SI')
  DX = 2B2Bh ('++')
Return: AL = FFh if installed
      CF clear
      ES:BX -> INT 29 entry point
      CX = ANSIPLUS BCD version number (v3.10+, CH=major, CL=minor)
      DL = capabilities (v4.00+)
    00h full capability driver
    01h reduced capability driver
    2Bh full capability driver (before v4.00)
Program: ANSIPLUS.SYS is a CON device driver by Kristofer Sweger which
    replaces the normal ANSI.SYS with a more powerful version having
    many additional features
Notes:  ANSIPLUS also identifies itself as ANSI.SYS if BX,CX, or DX differ
    from the magic values above
  an additional installation check is to test for the signature
    "ANSIPLUS" 12 bytes before the INT 29 entry point; the version
    number is also available as a four-character ASCII string (e.g.
    "4.00") four bytes before the entry point
SeeAlso: AX=1AA5h,AX=1AA6h,AX=1AA7h,AX=1AA8h,AX=1AA9h,AX=1AAAh,AX=D44Fh
--------V-2F1A00BX4156-----------------------
INT 2F - AVATAR.SYS - INSTALLATION CHECK
  AX = 1A00h
  BX = 4156h ('AV')
  CX = 4154h ('AT')
  DX = 4152h ('AR')
Return: AL = FFh if installed
      CF clear
      BX = AVATAR protocol level supported
      CX = driver type
    0000h AVATAR.SYS
    4456h DVAVATAR.COM inside DESQview window
      DX = 0016h
Program: AVATAR.SYS is a CON replacement by George Adam Stanislav which
    interprets AVATAR command codes in the same way that ANSI interprets
    ANSI command codes
Notes:  AVATAR also identifies itself as ANSI.SYS if BX, CX, or DX differ from
    the magic values
SeeAlso: AX=1A21h,AX=1A3Ch,AX=1A3Fh,AX=1A52h,AX=1A72h,AX=1A7Dh,AX=1AADh"AVATAR"
--------V-2F1A01-----------------------------
INT 2F U - DOS 4.0+ ANSI.SYS internal - GET/SET DISPLAY INFORMATION
```

```
  AX = 1A01h
  CL = function
       7Fh for GET
       5Fh for SET
  DS:DX -> parm block as for INT 21,AX=440Ch,CX=037Fh/035Fh respectively
Return: CF clear if successful
       AX destroyed
  CF set on error
       AX = error code (many non-standard)
Note: presumably this is the DOS IOCTL interface to ANSI.SYS
SeeAlso: AX=1A02h,INT 21/AX=440Ch
--------V-2F1A02-----------------------------
INT 2F U - DOS 4.0+ ANSI.SYS internal - MISCELLANEOUS REQUESTS
  AX = 1A02h
  DS:DX -> parameter block (see #02735)
Return: CF clear if successful
  CF set on error
       AX = error code
Note: DOS 5+ chains to previous handler if AL > 02h on call
SeeAlso: AX=1A01h

Format of ANSI.SYS parameter block:
Offset  Size  Description (Table 02735)
 00h  BYTE  subfunction
    00h set/reset interlock
    01h get /L flag
 01h  BYTE  interlock state
    00h=reset, 01h=set
      This interlock prevents some of the ANSI.SYS post-processing
      in its hook onto INT 10, AH=00h mode set
 02h  BYTE  (returned)
    00h if /L not in effect
    01h if /L in effect
--------V-2F1A21-----------------------------
INT 2F - AVATAR.SYS - SET DRIVER STATE
  AX = 1A21h (AL='!')
  DS:SI -> command string with one or more state characters (see #02736)
  CX = length of command string
Return: CF set on error (invalid subfunction)
  CF clear if successful
Note: the characters in the state string are interpreted left to right, and
```

```
    need not be in any particular order
SeeAlso: AX=1A00h/BX=4156h,AX=1A3Fh

(Table 02736)
Values for AVATAR.SYS state characters:
 'a'  activate driver
 'd'  disable driver
 'f'  use fast screen output
 'g'  always convert gray keys (+ and -) to function keys
 'G'  never convert gray keys
 'l'  convert gray keys only when ScrollLock active
 's'  use slow screen output
 't'  Tandy 1000 keyboard (not yet implemented)
--------V-2F1A3C-----------------------------
INT 2F U - AVATAR.SYS v0.11 - ???
  AX = 1A3Ch
  ???
Return: CX = 0000h
SeeAlso: AX=1A00h/BX=4156h,AX=1A21h,AX=1A3Eh
--------V-2F1A3E-----------------------------
INT 2F U - AVATAR.SYS v0.11 - ???
  AX = 1A3Eh
  CL = ???
  CH = ???
  DL = ???
  DH = ???
Return: CL = ???
  CH = ???
  DL = ???
  DH = ???
SeeAlso: AX=1A3Ch,AX=1A3Fh
--------V-2F1A3F-----------------------------
INT 2F - AVATAR.SYS - QUERY DRIVER STATE
  AX = 1A3Fh (AL='?')
  ES:DI -> buffer
  CX = length of buffer in bytes
Return: CF clear
  CX = actual size of returned info
Note: the returned information consists of multiple letters whose meanings
    are described under AX=1A21h
SeeAlso: AX=1A00h/BX=4156h,AX=1A21h,AX=1A44h
```

```
--------S-2F1A42BX4156-----------------------
INT 2F - AVATAR Serial Dispatcher - INSTALL IRQ3 HANDLER
  AX = 1A42h
  BX = 4156h ('AV')
  ES:DI -> FAR handler for serial port using IRQ3
  DS = data segment needed by handler
Return: AX = status/return value
      0000h if no more room
      1A42h if ASD not installed
      else handle to use when uninstalling
Notes:  the handler need not save/restore registers or signal EOI to the
    interrupt controller
  the handler should return AX=0000h if the interrupt was meant for it,
    and either leave AX unchanged or return a non-zero value otherwise
  the most recently installed handler will be called first, continuing
    to earlier handlers until one returns AX=0000h
SeeAlso: AX=1A43h,AX=1A62h
--------S-2F1A43BX4156-----------------------
INT 2F - AVATAR Serial Dispatcher - INSTALL IRQ4 HANDLER
  AX = 1A43h
  BX = 4156h ('AV')
  ES:DI -> FAR handler for serial port using IRQ4
  DS = data segment needed by handler
Return: AX = status/return value
      0000h if no more room
      1A43h if ASD not installed
      else handle to use when uninstalling
Notes:  (see AX=1A42h)
SeeAlso: AX=1A42h,AX=1A63h
--------V-2F1A44BX4156-----------------------
INT 2F - AVATAR.SYS v0.11+ - GET DATA SEGMENT
  AX = 1A44h
  BX = 4156h ('AV')
Return: AX = 0000h
  DS = data segment
  CX = size of data segment
Note: AVATAR.SYS calls this function whenever it is invoked.  If each
    process under a multitasker hooks this function and provides a
    separate data segment, AVATAR.SYS becomes fully reentrant.
SeeAlso: AX=1A21h,AX=1A3Fh,AX=1A52h
--------V-2F1A52----------------------------
```

```
INT 2F U - AVATAR.SYS v0.11 - GET ???
  AX = 1A52h
  CX = size of buffer
  ES:DI -> buffer
Return: ??? copied into user buffer
Note: the maximum size of the data which may be copied is returned by
    AX=1A72h
SeeAlso: AX=1A53h,AX=1A72h
--------V-2F1A53----------------------------
INT 2F U - AVATAR.SYS v0.11 - ???
  AX = 1A53h
  CL = ??? (00h-05h)
  ???
Return: ???
SeeAlso: AX=1A00h/BX=4156h,AX=1A52h,AX=1A72h
--------S-2F1A62BX4156----------------------
INT 2F - AVATAR Serial Dispatcher - UNINSTALL IRQ3 HANDLER
  AX = 1A62h
  BX = 4156h ('AV')
  CX = handle for IRQ routine returned by AX=1A42h
SeeAlso: AX=1A42h,AX=1A63h
--------S-2F1A63BX4156----------------------
INT 2F - AVATAR Serial Dispatcher - UNINSTALL IRQ4 HANDLER
  AX = 1A63h
  BX = 4156h ('AV')
  CX = handle for IRQ routine returned by AX=1A43h
SeeAlso: AX=1A43h,AX=1A62h
--------V-2F1A72----------------------------
INT 2F U - AVATAR.SYS v0.11 - GET ??? SIZE
  AX = 1A72h
Return: CX = maximum size of ???
SeeAlso: AX=1A00h/BX=4156h,AX=1A52h,AX=1A7Bh,AX=1AADh"AVATAR"
--------V-2F1A7B----------------------------
INT 2F U - AVATAR.SYS v0.11 - ???
  AX = 1A7Bh
Return: AX = 0000h
  CX = 0000h
SeeAlso: AX=1A00h/BX=4156h,AX=1A72h,AX=1A7Dh
--------V-2F1A7D----------------------------
INT 2F U - AVATAR.SYS v0.11 - ???
  AX = 1A7Dh
```

```
Return: AX = ???
SeeAlso: AX=1A00h/BX=4156h,AX=1A7Bh
--------V-2F1AA3-----------------------------
INT 2F - ANSIPLUS v4.03+ - GET/SET ANSIPLUS INTERNAL VARIABLES
  AX = 1AA3h
  BH = function
       00h get current/default colors
    Return: CH = default colors
      CL = current colors
      01h set current/default colors
    CH = default colors (00h = leave unchanged)
    CL = current colors
      02h get current subscreen region
    Return: BH,BL = true screen rows,columns
      CH,CL = top left row,column of region
      DH,DL = bottom right row,column of region
      03h set subscreen region
    CH,CL = top left row,column of region
    DH,DL = bottom right row,column of region
      04h get driver features (bits 0-31)
    Return: DX:CX = current feature bits
      05h set driver features (bits 0-31)
    DX:CX = feature bits
      06h get driver features (bits 32-63)
    Return: DX:CX = current feature bits
      07h set driver features (bits 32-63)
    DX:CX = feature bits
      other: reserved for future use
SeeAlso: AX=1AA4h,AX=1AA5h
--------V-2F1AA4-----------------------------
INT 2F - ANSIPLUS v4.02+ - GET/SET ANSIPLUS SMOOTH SCROLLING RATE
  AX = 1AA4h
  BL = function
       00h get scrolling rate
       01h set scrolling rate
    BH = new minimum scrolling rate in scan lines per retrace
Return: BH = smooth scrolling rate
SeeAlso: AX=1AA3h,AX=1AA5h
--------V-2F1AA5-----------------------------
INT 2F - ANSIPLUS v4.00+ - GET/SET ANSIPLUS CLIPBOARD
  AX = 1AA5h
```

```
  DH = subfunction
       00h get clipboard information
       01h get clipboard text
       02h set clipboard text
       03h append text to clipboard
       04h clear clipboard
       05h paste clipboard to keyboard
  ES:BX -> data area for subfunctions 01h, 02h, and 03h
  CX = size of data area (maximum size for subfunction 01h, actual size
       to add to clipboard for subfunctions 02h and 03h)
Return: AL = status
       00h successful
       01h unsupported subfunction (reduced capability driver)
       02h insufficient space
       A5h unsupported function (ANSIPLUS before v4.00)
  ES:BX -> ANSIPLUS local clipboard data
  CX = number of bytes currently in local clipboard
  DX = maximum size of local clipboard
SeeAlso: AX=1A00h/BX=414Eh,AX=1AA4h,AX=1AA6h
--------V-2F1AA6-----------------------------
INT 2F - ANSIPLUS v4.00+ - ENABLE/DISABLE ANSIPLUS DRIVER
  AX = 1AA6h
  BH = function
       00h get hooked interrupts
       01h set hooked interrupts mask
    BL = new interrupts mask (see #02737)
Return: BL = previous interrupts mask (see #02737)
SeeAlso: AX=1A00h/BX=414Eh,AX=1AA7h

Desc: used to temporarily disable any prior copies of ANSIPLUS when a new
    copy is installed, such as in a multitasking system like DESQview
Note: only the most-recently loaded copy of ANSIPLUS on the current INT 2F
    chain responds to this call

Bitfields for ANSIPLUS hooked interrupts mask:
Bit(s)  Description (Table 02737)
 0  INT 09 hook disabled
 1  INT 10 hook disabled
 2  INT 15 hook disabled
 3  INT 16 hook disabled
 4  INT 1C hook disabled
```

```
 5  reset all bits when INT 29 called
 6  INT 29 hook disabled
 7  INT 33, INT 74, or other mouse event hook disabled
--------V-2F1AA7-----------------------------
INT 2F - ANSIPLUS v4.00+ - ENABLE/DISABLE ANSIPLUS FEATURES
  AX = 1AA7h
  BL = function
       00h prevent scroll-back saves
       01h enable scroll-back saves
       02h disable key reprogramming and lock changes by escape sequences
       03h enable key reprogramming by escape sequences
       04h disable and lock key stacking changes by escape sequences
       05h allow key stacking by escape sequences
Return: nothing
SeeAlso: AX=1AA6h
--------V-2F1AA8-----------------------------
INT 2F - ANSIPLUS v3.10+ - GET NEXT ANSIPLUS SCROLLBACK LINE
  AX = 1AA8h
Return: AL = status
       00h successful
    ES:BX -> screen line (character and attribute pairs)
    CX = length of line in bytes, 0000h if no more lines or
           unsupported video mode
       01h unsupported video mode active
       02h screen currently scrolled back
       03h reduced capability driver
       A8h unsupported function (driver before v3.10)
SeeAlso: AX=1A00h/BX=414Eh,AX=1AA9h
--------V-2F1AA9-----------------------------
INT 2F - ANSIPLUS v3.10+ - GET ANSIPLUS SCROLLBACK INFORMATION
  AX = 1AA9h
Return: AL = status
       00h successful
    BX = current number of lines in scrollback buffer
    CX = number of bytes in one line
       01h unsupported video mode active
       02h screen currently scrolled back
       03h reduced capability driver
       A9h unsupported function (driver before v3.10)
Desc: determine how much data is in the scrollback buffer and initialize
    scrollback retrieval to return the first line on the next call to
```

```
    AX=1AA8h
SeeAlso: AX=1A00h/BX=414Eh,AX=1AA8h
--------V-2F1AAA----------------------------
INT 2F - ANSIPLUS v3.01+ - GET/SET ANSIPLUS SCREEN SAVER BLANKING TIME
  AX = 1AAAh
  BX = function
       FFFFh to get current blanking time
       other to set time
    CX = blanking time in clock ticks (0000h-7FFFh)
Return: BX = current blanking time
  CX = blanking time when last set
SeeAlso: AX=1A00h/BX=414Eh,AX=1AABh
--------V-2F1AAB----------------------------
INT 2F - ANSIPLUS v3.01+ - SET ANSIPLUS KEY REPEAT RATE
  AX = 1AABh
  BX = repeat rate in characters per second
       0000h use BIOS repeat rate
Return: nothing
SeeAlso: AX=1A00h/BX=414Eh,AX=1AAAh,AX=1AACh
--------V-2F1AAC----------------------------
INT 2F - ANSIPLUS v3.00+ - LOAD CHARACTER GENERATOR
  AX = 1AACh
  BH = number of bytes per character pattern
  BL = VGA/EGA character table to be loaded
  CX = number of characters to load
  DX = starting character code (offset into Map2 block)
  ES:BP -> user character table to be loaded
Return: AX = 1100h
Desc: load the EGA/VGA character generator without the BIOS function's
    side effects of resetting the video mode and color palette
SeeAlso: AX=1A00h/BX=414Eh,AX=1AABh,AX=1AADh"ANSIPLUS",INT 10/AX=1100h
--------V-2F1AAD----------------------------
INT 2F - ANSIPLUS v2.00+ - ANSIPLUS DEVICE STATUS REPORT
  AX = 1AADh
  BL = report request code (81h-96h for v4.00)
  CX = color selector or key code, if required by request
Return: AX = first reported result
  BX = second result
  CX = third result, if applicable (unchanged otherwise)
  DX = fourth result, if applicable (unchanged otherwise)
Desc: get device status reports equivalent to those for Esc [#n sequences
```

```
    while bypassing any device redirection and avoiding the need to
    parse the returned result
Note: the report request code in BL is identical to the number in the
    corresponding Esc [#n sequence
SeeAlso: AX=1A00h/BX=414Eh,AX=1AACh
--------V-2F1AADDX0000----------------------
INT 2F U - AVATAR.SYS v0.11 - ???
  AX = 1AADh
  DX = 0000h
  CX = subfunction (00h-0Ch)
  ???
Return: AX = 0000h if DX was nonzero
  ???
SeeAlso: AX=1A00h/BX=4156h,AX=1A72h
--------m-2F1B00-----------------------------
INT 2F U - DOS 4+ XMA2EMS.SYS extension internal - INSTALLATION CHECK
  AX = 1B00h
Return: AL = FFh if installed
Note: XMA2EMS.SYS extension is only installed if DOS has page frames to hide.
  This extension hooks onto INT 67/AH=58h and returns from that call data
    which excludes the physical pages being used by DOS.
SeeAlso: AH=1Bh"FRAME INFO"
--------m-2F1B-------------------------------
INT 2F U - DOS 4+ XMA2EMS.SYS extension internal - GET HIDDEN FRAME INFORMATION
  AH = 1Bh
  AL <> 00h
  DI = hidden physical page number
Return: AX = FFFFh if failed (no such hidden page)
  AX = 0000h if OK, then
       ES = segment of page frame
       DI = physical page number
Notes:  this corresponds to the data edited out of the INT 67/AH=58h call
  FASTOPEN makes this call with AL = FFh
SeeAlso: AX=1B00h
--------V-2F2300-----------------------------
INT 2F - DR DOS 5.0 GRAFTABL - INSTALLATION CHECK
  AX = 2300h
Return: AH = FFh
Note: this installation check does not follow the usual format
SeeAlso: AH=23h,AX=2E00h
--------V-2F23-------------------------------
```

```
INT 2F - DR DOS 5.0 GRAFTABL - GET GRAPHICS DATA
  AH = 23h
  AL nonzero
Return: AH = FFh
  ES:BX -> graphics data (8 bytes for each character from 80h to FFh)
SeeAlso: AX=2300h,AX=2E00h
--------T-2F2700-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - INSTALLATION CHECK
  AX = 2700h
Return: AL = status
      00h not installed
      FFh installed
Note: the TaskMAX API is also supported by Novell DOS 7 TASKMGR in both
    taskswitching and multitasking modes
--------T-2F2701-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - GET STATUS
  AX = 2701h
Return: AX = maximum simultaneous tasks
  BX = index into TASK_IDS of current foreground task
  CX = currently-active tasks
  DX = version number (DL = major, DH = minor)
     (DR DOS 6.0 = 0001h, Novell DOS 7 = 0002h)
  ES:SI -> TASK_IDS
  ES:DI -> name table (array of 8-byte names, NUL-terminated if <8 chars)
Notes:  do not attempt to create a new task if CX == AX
  the task's index is its position on the task menu, while its ID is the
    position within the internal task name table
SeeAlso: AX=2714h,AX=2716h
--------T-2F2702-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - GET PER-TASK EMS LIMIT
  AX = 2702h
Return: DX = maximum pages INT 67/AH=42h will report available
Note: TaskMAX does not limit EMS allocations other than by limiting the
    amount which is reported as being available at a given time
SeeAlso: AX=2703h,INT 67/AH=42h
--------T-2F2703-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - SET PER-TASK EMS LIMIT
  AX = 2703h
  DX = maximum pages INT 67/AH=42h should report available
Return: DX = new maximum for reporting
Note: the TaskMAX API is also supported by Novell DOS 7 TASKMGR in
```

```
    both taskswitching and multitasking modes
SeeAlso: AX=2702h,INT 67/AH=42h
--------T-2F2704-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - REGISTER/UNREGISTER TASK MANAGER
  AX = 2704h
  DL = subfunction
       00h unregister task manager
       01h register task manager
Return: DL = status
       00h registered
       01h unregistered
Notes:  a task manager replaces TaskMAX's menu system with its own user
    interface; while one is registered, the TaskMAX hotkeys and
    Ctrl-Alt-Del invoke the manager rather than the built-in menu system
  unregister the task manager before terminating it
SeeAlso: AX=2705h
Index:  hotkeys;TaskMAX
--------T-2F2705-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - ENABLE/DISABLE DIRECT SWITCHING
  AX = 2705h
  DL = subfunction
       00h disable keystrokes for switching to next/prev/specified task
       01h enable
Return: nothing
Note: should only be called by a registered task manager (see AX=2704h)
SeeAlso: AX=2704h,AX=2706h
--------T-2F2706-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - SWITCH TO SPECIFIED TASK
  AX = 2706h
  DX = task index (see AX=2701h) of task to be activated
Return: DX = task index of previously-active task
Note: the TaskMAX API is also supported by Novell DOS 7 TASKMGR in
    both taskswitching and multitasking modes
SeeAlso: AX=2705h,AX=2707h,AX=2715h
--------T-2F2707-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - CREATE NEW TASK
  AX = 2707h
  DS:DX -> ASCIZ pathname of executable
  ES:BX -> parameter block (see #02738)
  CX = number of ticks before automatic return to task manager
    (0000h = run until termination or explicitly switched)
```

```
Return: DX = new task's task index (FFFFh if task terminated)
SeeAlso: AX=2706h,AX=2708h

Format of TaskMAX parameter block:
Offset  Size  Description (Table 02738)
 00h  WORD  reserved, should be 0000h
 02h  DWORD pointer to command tail to be copied into child's PSP
 06h  DWORD pointer to first FCB to be copied into child's PSP
 0Ah  DWORD pointer to second FCB to be copied into child's PSP
--------T-2F2708-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - DELETE TASK
  AX = 2708h
  DX = task index
Return: DX = FFFFh (task deleted)
Notes:  this call should only be used for abnormal task termination, after
    first checking for open files with AX=270Ch; should not be used
    with programs that allocate EMS or XMS memory
  switches to specified task first
SeeAlso: AX=2707h
--------T-2F2709-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - NAME TASK
  AX = 2709h
  DX = task index
  DS:SI -> 8-byte name (8 NULs = remove name)
Return: AL = task flags
      00h ID unused or task terminated
      01h ID in use, task name table entry valid
      81h ID in use, task name fixed
  BX = task ID
  ES:DI -> name in task name table (see AX=2701h)
Note: the task retains the given name until it terminates or the name is
    removed by specifying a name of 8 NULs.
SeeAlso: AX=2701h,AX=2707h
--------T-2F270A-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - CONVERT TASK INDEX TO TASK ID
  AX = 270Ah
  DX = task index
Return: DX = task ID (FFFFh if index invalid)
Note: task IDs stay constant, while indexes can change when other tasks are
    deleted
SeeAlso: AX=2701h,AX=270Bh
```

```
--------T-2F270B-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - CONVERT TASK ID TO TASK INDEX
  AX = 270Bh
  DX = task ID
Return: DX = task index (FFFFh if task not active)
Note: the TaskMAX API is also supported by Novell DOS 7 TASKMGR in both
    taskswitching and multitasking modes
SeeAlso: AX=270Ah
--------T-2F270C-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - CHECK OPEN FILES
  AX = 270Ch
  DX = task index
Return: AX = number of files currently open for specified task
SeeAlso: AX=2708h
--------T-2F270D-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - CHECK IF TASK RUNNING PRIMARY COMMAND INTERPRETER
  AX = 270Dh
  DX = task index
Return: DX = status
      0000h if primary command interpreter (COMMAND.COM, etc.) running
      0001h if not in root shell for task
Note: TaskMAX will return 0001h if the specified task has spawned another
    command interpreter with AX=2707h
SeeAlso: AX=2707h,AX=270Ch
--------T-2F270E-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - GET/SET TEXT PASTE LEAD-IN
  AX = 270Eh
  CX = length of string (max 15 keystrokes, 0000h to get current string)
  DS:SI -> pasting lead-in string (character/scan-code pairs)
Return: ES:DI -> current lead-in string
Note: the specified sequence of keystrokes is sent to the application before
    every line of a text-mode spreadsheet paste
SeeAlso: AX=270Fh,AX=2710h,AX=2713h
--------T-2F270F-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - GET/SET NUMERIC PASTE LEAD-IN
  AX = 270Fh
  CX = length of string (max 15 keystrokes, 0000h to get current string)
  DS:SI -> pasting lead-in string (character/scan-code pairs)
Return: ES:DI -> current lead-in string
Note: the specified sequence of keystrokes is sent to the application before
    every number in a numeric-mode spreadsheet paste
```

```
SeeAlso: AX=270Eh,AX=2710h,AX=2711h,AX=2713h
--------T-2F2710-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - GET/SET PASTE LINE TERMINATOR STRING
  AX = 2710h
  CX = length of string (max 15 keystrokes, 0000h to get current string)
  DS:SI -> pasting terminator string (character/scan-code pairs)
Return: ES:DI -> current terminator string
Note: the specified sequence of keystrokes is sent to the application after
    every line of a spreadsheet paste operation
SeeAlso: AX=270Eh,AX=270Fh,AX=2713h
--------T-2F2711-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - GET/SET NUMERIC PASTE DECIMAL POINT
  AX = 2711h
  DX = ASCII code for separator (FFFFh to get current)
Return: DL = current separator character
SeeAlso: AX=270Fh
--------T-2F2712-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - INITIATE EXPORTING TASK DATA
  AX = 2712h
  DX = task index
--------T-2F2713-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - INITIATE PASTE OPERATION
  AX = 2713h
  DX = task index
  CX = paste mode
      0000h alphanumeric
      0001h numeric
      0002h text
SeeAlso: AX=270Eh,AX=270Fh,AX=2710h,AX=2711h
--------T-2F2714-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - GET SWAP SPACE INFO
  AX = 2714h
Return: CX = total KB of swap space
  DX = available KB of swap space
Note: the TaskMAX API is also supported by Novell DOS 7 TASKMGR in both
    taskswitching and multitasking modes
SeeAlso: AX=2701h
--------T-2F2715-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - SWITCH TO TASK MANAGER
  AX = 2715h
Return: only after calling task is again selected
```

```
SeeAlso: AX=2706h
--------T-2F2716-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - GET PASTE BUFFER STATUS
  AX = 2716h
Return: AX = 0000h if AX=2716h,AX=2717h,AX=2718h supported
      CX = bytes in paste buffer
      DX = current generation number (updated after every copy operation)
BUG:  Novell DOS 7 TASKMGR returns AX=0000h even though it does not support
    this call (it does support the remainder of the TaskMAX API)
SeeAlso: AX=2701h,AX=2713h,AX=2714h,AX=2717h,AX=2718h
--------T-2F2717-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - PASTE DATA DIRECTLY TO APPLICATION BUFFER
  AX = 2717h
  CX = bytes in destination buffer
  ES:DI -> destination buffer
Return: AX = 0000h if function supported
      CX = bytes actually copied (FFFFh if buffer too small)
      DX = current generation number for paste buffer
BUG:  Novell DOS 7 TASKMGR returns AX=0000h even though it does not support
    this call (it does support the remainder of the TaskMAX API)
Note: the destination buffer may be too small if another task adds more data
    to the paste buffer after the AX=2716h call but before this call
SeeAlso: AX=2713h,AX=2716h,AX=2718h
--------T-2F2718-----------------------------
INT 2F - DR DOS 6.0 TaskMAX - COPY DATA DIRECTLY INTO PASTE BUFFER
  AX = 2718h
  CX = bytes in source buffer
  DS:SI -> source buffer (plain ASCII, lines terminated with CR LF)
Return: AX = 0000h if function supported
      CX = bytes actually copied
      DX = current generation number for paste buffer
BUG:  Novell DOS 7 TASKMGR returns AX=0000h even though it does not support
    this call (it does support the remainder of the TaskMAX API)
SeeAlso: AX=2712h,AX=2716h,AX=2717h
--------T-2F2719-----------------------------
INT 2F - Novell DOS 7 TaskMGR - NOP
  AX = 2719h to 271Bh
--------T-2F271C-----------------------------
INT 2F U - Novell DOS 7 TaskMGR - ???
  AX = 271Ch
  DX = ???
```

```
      bit 0: ???
Return: ???
  ---if DX bit 0 set---
  AX = 0031h
  CX = 0000h
BUG:  if the task switcher is running, and DX bit 0 is set on call, this
    function will crash because its exit code attempts to pop several
    registers which are not pushed when DX bit 0 is set
--------m-2F2780CL01-------------------------
INT 2F U - Novell DOS 7 - EMM386.EXE - MULTITASKING API - SUPERVISOR MODULE
  AX = 2780h
  CL = 01h (module: Supervisor)
  CH = subfunction
      00h unused
    Return: CX = status (0002h) (see #02739)
      01h unused
    Return: CX = status (0002h) (see #02739)
      02h "Z_ModuleReg" register an OS module
    EBX = module number (0005h-000Fh)
    DS:EDX -> module descriptor (see #02741)
    Return: CX = status (0000h,0030h) (see #02739)
      03h allocate SYSDAT memory
    DX = number of ??? to allocate
    Return: CX = status (0000h,0003h) (see #02739)
      04h get selector to SYSDAT
    Return: CX = 0000h (successful)
      BX = selector for EMM386 data segment
      EBX high word cleared
      05h "Z_MoveReal" relocate segment into extended memory
    DS:EDX -> descriptor parameter block
    Return: ECX = status (00h,03h,31h,32h) (see #02739)
      ---if successful---
      parameter block filled
      06h "Z_Reboot" return to real mode via triple fault
      07h debugger break
    Note: calls INT 03, then INT 21/AH=02h to output a question
        mark
      08h "X_ForeCheck" check if domain is in foreground
    Return: CX = 0000h (successful)
      EBX = ??? (0 or 2)
      09h register VxD with system
```

```
  0Ah unload VxD hook
  0Bh indicate end of initialization phase
Return: CX = status (0002h) (see #02739)
  0Ch "F_AllocWindow" allocate 4K mapping window
  0Dh "F_RegisterBoot" register reboot addresses
EBX = ???
EDX = ???
Return: ???
  0Eh "F_EnquireBoot" check if reboot active
Return: CX = 0000h (successful)
  BL = ???     \ or BX = 0000h
  BH = ???     /
  0Fh get debugging level
Return: CX = 0000h (successful)
  EBX = new value for debugging level
  10h set debugging level
EDX = ???
Return: CX = 0000h (successful)
  EBX = old value of debugging level
  11h installation check (documented)
Return: CX = status
      0000h if multitasker is installed
    EBX = version (0100h for v1.00)
      1101h if multitasker is not present
  12h "F_V86BPInstall" install V86 breakpoint
DX = ???
Return: CX = 0000h (successful)
  AX = old value of ???
  13h "F_V86BPRemove" remove V86 breakpoint
Return: CX = status (0000h,003Fh) (see #02739)
  AX = ???
  14h "F_V86BPOffer" indicate INT 03 to be used as V86 breakpoint
EDX = linear address ??? of INT 03 instruction for breakpoint
Return: CX = status (0000h,003Fh) (see #02739)
  15h "F_LoaderCleanup" offer opportunity to clean up
BX = segment of ???
Return: CX = 0000h (successful)
  BX = segment of ???
  16h "F_RegisterVxDWindow" register VxD mapping window
  17h "F_RegisterPNW" register Personal NetWare information
EBX = subfunction (0-2)
```

```
    Return: CX = status (0002h if EBX>2) (see #02739)
      ???
      18h unused
    Return: CX = status (0002h) (see #02739)
Return: CX = status (most subfunctions)
  (E)AX and/or (E)BX contain return values, depending on function
Notes:  called by DPMS.EXE and EMM386.EXE
  this API is only available if AX=12FFh/BX=0EDCh returns successfully;
    because the request is handled on the initial trap to the memory
    manager caused by INT instructions, this API must be invoked with
    an actual INT 2F instruction instead of some simulation such as a
    far call to the address in the interrupt vector table
SeeAlso: AX=12FFh/BX=0EDCh,AX=2780h/CL=02h,AX=2780h/CL=03h,AX=2780h/CL=04h

(Table 02739)
Values for Novell DOS 7 EMM386 function status:
 0000h  successful
 0001h  not implemented
 0002h  invalid subfunction
 0003h  unable to find memory
 0004h  invalid flag (semaphore) number
 0005h  flag (semaphore) overrun
 0006h  flag (semaphore) underrun
 0007h  no queue handles available
 0009h  no queue buffers available
 000Ah  queue is in use
 000Bh  invalid process handle
 000Ch  no process handles available
 000Dh  queue access not permitted
 000Eh  queue is empty
 000Fh  queue is full
 0012h  no memory handles available
 0014h  can't find process in process list
 001Bh  invalid memory handle
 0023h  unable to terminate process
 002Ah  flag set ignored
 002Dh  no more system flags
 002Eh  flag (semaphore) not in idle state
 002Fh  flag (semaphore) wait timed out
 0030h  bad module number in CL
 0031h  bad descriptor
```

```
 invalid value for DESC_PB_SINFO in a descriptor parameter block
0032h  no free descriptors
0033h  error while locking/unlocking a page
0034h  error getting or setting a page
0035h  no pages available
0036h  invalid domain
0037h  process already frozen
0038h  process not frozen
0039h  fork failure (no registered swaplist)
003Ah  page already free
003Bh  page already allocated
003Ch  unable to switch tasks
003Dh  attempted to free critical section which is not active
003Eh  too many active critical sections
003Fh  ???
41FFh  current domain is invalid, or no domain in context
42FFh  domain ID is not a valid descriptor
43FFh  domain creation still in progress
44FFh  domain currently being deleted
45FFh  task manager is busy, cannot unload it
46FFh  task manager already loaded
47FFh  task manager not yet loaded
48FFh  cannot save/restore because prior switch not complete
49FFh  console already has owner
4BFFh  unsupported opcode
4CFFh  32-bit address prefix not supported
4FFFh  timeout, but not on timer queue
50FFh  unable to lock timer queue
53FFh  unable to switch while in Global Message Mode
54FFh  error while setting Global Message Mode
55FFh  not in Global Message Mode
56FFh  system already in Domain Message Mode
57FFh  not in Domain Message Mode
58FFh  unable to allocate timeout structure
59FFh  unsupported video type
5AFFh  function not handled by any VM
5BFFh  error in Serial..Set call
5CFFh  error in Parallel..Set call
5DFFh  domain list overflowed
5FFFh  unable to free domain while in nobuffers mode
```

```
Format of Novell DOS/OpenDOS EMM386 descriptor parameter block:
Offset  Size  Description (Table 02740)
 00h  DWORD "DESC_PB_BASE"
 04h  DWORD "DESC_PB_LIMIT"
 08h  WORD  "DESC_PB_SEL"
 0Ah  BYTE  "DESC_PB_MINFO"
 0Bh  BYTE  "DESC_PB_SINFO"
SeeAlso: #02741

Format of Novell DOS/OpenDOS EMM386 module descriptor:
Offset  Size  Description (Table 02741)
 00h 12 BYTEs descriptor parameter block (see #02740)
 0Ch  DWORD -> module entry point
Note: for function "Z_ModuleReg", only DESC_PB_SEL and the module entry point
    need to be initialized before calling EMM386
SeeAlso: #02740
--------m-2F2780CL02-------------------------
INT 2F U - Novell DOS 7 - EMM386.EXE - MULTITASKING API - REAL-TIME MONITOR
  AX = 2780h
  CL = 02h (module: Real-Time Monitor)
  CH = subfunction
      00h ???
    ???
    Return: CX = status (0000h,0023h, others???) (see #02739)
      BX = ??? (0000h if CX=0000h, FFFFh if CX=0023h)
    Note: calls func 04h/sf 03h, func 04h/sf 1Eh,func 02h/sf 43h,
        func 02h/sf 0Bh, and func 04h/sf 01h
      01h "Z_PCountGet" get and clear count of dispatches
    DX = ??? handle or 0000h for default
    Return: CX = status (0000h,000Bh) (see #02739)
      EBX = old value of ??? if successful
      EDX destroyed
      02h "Z_FlagWait" wait on semaphore
    DX = index of semaphore???
    Return: CX = status (0000h,0004h,0005h) (see #02739)
      EBX = 0000FFFFh on error, 00000000h if successful
      03h "Z_FlagSet" set a semaphore flag
    DX = index of semaphore???
    Return: CX = status (0000h,0004h,0006h) (see #02739)
      BX = FFFFh on error, 0000h if successful
      04h "X_QCreate" create a new queue
```

```
Return: CX = status (0000h,0007h,000Ah) (see #02739)
  05h "X_QOpen" open queue for reading/writing
EDX -> ??? data (first 8 bytes seem to be name)
Return: CX = status (0000h,0009h,000Dh) (see #02739)
  06h "X_QDelete" delete a queue
Note: calls fn 02h/subfn 05h, then fn 02h/subfn 40h
  07h "X_QRead" read message from queue
EDX = ???
Return: CX = status (0000h,0009h,000Eh) (see #02739)
  08h "X_QReadC" read message from queue, if any
EDX = ???
Return: CX = status (0000h,0009h,000Eh) (see #02739)
  09h "X_QWrite" write message to queue
EDX = ???
Return: CX = status (0000h,0009h,000Fh) (see #02739)
  0Ah "X_QWriteC" write message to queue, if space available
EDX = ???
Return: CX = status (0000h,0009h,000Fh) (see #02739)
  0Bh "X_PDelay" put process to sleep for specified period
DX = number of clock ticks to sleep???
Return: CX = 0000h (successful)
  0Ch "X_PDispatch" force a dispatch (run scheduler)
Return: CX = 0000h (successful)
  0Dh "F_PTerm" terminate process???
Return: CX = status (0000h,0023h) (see #02739)
  BX = FFFFh on error, 0000h if successful
Note: calls fn 04h/subfn 03h, fn 04h/subfn 1Eh,
    fn 02h/subfn 43h, fn 02h/subfn 0Bh, fn 04h/sub 01h
  0Eh "X_PCreate" create new process
???
Return: CX = status (0000h,000Ch) (see #02739)
  EBX = ??? if successful
  0Fh "Z_PPriorSet" set process priority
BX = ???
DX = ??? handle or 0000h for default
Return: CX = status (0000h,000Bh) (see #02739)
  10h "X_PHandleGet" get current process handle
Return: CX = 0000h (successful)
  BX = handle of default ???
  EBX high word cleared
  11h "X_PTerm" terminate process
```

```
DX = process handle or 0000h for current
BX = ??? (handle???)
Return: CX = status (0000h,000Bh,0014h) (see #02739)
  12h "F_Sleep" ???
BX = ???
DX = ???
Return: CX = 0000h (successful)
  13h "F_Wakeup" ???
DX = ???
Return: CX = status (see #02739)
  14h "F_FindPDName" find process by name???
BX = ???
DX = ???
Return: CX = status (0000h,0014h) (see #02739)
  BX = FFFFh on error, ??? if successful
  15h "F_SetFlags" set ??? flags
BX = ??? (low two bits only)
DX = ??? handle or 0000h for default
Return: CX = status (0000h,000Bh) (see #02739)
  BX = new value of ??? flags (entire word)
  EBX high register cleared
  16h "F_EndOfInterrupt" issue EOI to PIC
Return: CX = 0000h (successful)
  17h "X_PTermOff" disable process termination
  18h "X_PTermOK" enable process termination
  19h "Z_FlagStatusGet" get semahprore's status
  1Ah "F_QRdMX" ???
  1Bh "F_QWrMX" ???
  1Ch "Z_FlagAlloc" allocate a new semaphore
  1Dh "Z_FlagFree" free semaphore
  1Eh "X_FlagsMaxGet" get number of semaphores supported
  1Fh "X_QReadNDC" non-destructive read from queue (peek)
  20h "Z_FlagWWTO" wait on semaphore, with timeout
  21h "F_UdaAlloc" ???
  22h "F_UdaFree" ???
  23h "X_PSuspend" suspend process
  24h "X_PUnsuspend" restart process after suspension
  25h "X_CritEnter" enter critical region
  26h "X_CritExit" leave critical region
  27h "F_PCreate" ???
  28h "Z_PHandleListGet" get list of process handles
```

```
      29h "Z_PNameGet" get process name
      2Ah "Z_PStatusGet" get process status
      2Bh "F_PDToDomain" get process' domain???
      2Ch "Z_PPriorGet" get process priority
      2Dh "F_QDList" get list of queues???
      2Eh "Z_QNameGet" get queue's name
      2Fh "X_QMsgLenGet" get message length of queue
      30h "X_QMsgMaxGet" get message capacity of queue
      31h "Z_QWriterGet" get handle of process waiting to write queue
      32h "Z_QReaderGet" get handle of process waiting to read queue
      33h "X_QMsgNumGet" get number of messages in queue
      34h "Z_QFlagsGet" get queue's flags
      35h "F_NameToQD" get queue by name???
      36h "F_NameToPD" get process by name???
      37h "X_MXCreate" create a mutex
      38h "X_MXDelete" delete a mutex
      39h "X_MXEnter" enter mutual-exclusion zone
      3Ah "X_MXEnterC" enter mutual-exclusion zone if it is free
      3Bh "X_MXExit" leave mutual-exclusion zone
      3Ch "Z_TicksSet" set length of foreground time slices
      3Dh "X_TickGet" get clock tick period
      3Eh "F_ProcessID" ???
      3Fh "X_QClose" close a queue
      40h "F_QDispose" delete queue???
      41h "F_PDToFlags" get process flags???
      42h "F_PDToDParam" ???
      43h "F_ReleaseMX" release mutex???
      44h "F_SimulateInt" ???
      45h "Z_QFlagsSet" set queue's flags
      46h "F_TickRate" ???
      47h ???
Return: CX = status (most subfunctions)
  (E)AX and/or (E)BX contain return values, depending on function
Notes:  called by DPMS.EXE and EMM386.EXE
  this API is only available if AX=12FFh/BX=0EDCh returns successfully;
    because the request is handled on the initial trap to the memory
    manager caused by INT instructions, this API must be invoked with
    an actual INT 2F instruction instead of some simulation such as a
    far call to the address in the interrupt vector table
SeeAlso: AX=12FFh/BX=0EDCh,AX=2780h/CL=01h,AX=2780h/CL=03h,AX=2780h/CL=04h
--------m-2F2780CL03-------------------------
```

```
INT 2F U - Novell DOS 7 - EMM386.EXE - MULTITASKING API - MEMORY
  AX = 2780h
  CL = 03h (module: Memory)
  CH = subfunction
      00h ???
      01h "F_PdirGet" ???
      02h "F_PdirSet" ???
      03h "Z_PageAlloc" allocate a given page of memory
      04h "F_PageAllocG" allocate global page???
      05h "F_PageAllocI" allocate instance page???
      06h "F_PtblRead" get page table???
      07h "F_PtrlWrite" select page table???
      08h "Z_PtblGet" read page table
      09h "Z_PtblSet" change page table
      0Ah "Z_PagesAlloc" allocate pages of memory
      0Bh "Z_PageFree" free a given page of memory
      0Ch "Z_MemAlloc" allocate a memory block
      0Dh "Z_MemFree" release memory block
      0Eh "Z_MemSizeGet" get size of memory block
      0Fh "Z_MemResize" resize a memory block
      10h "Z_DescAlloc" allocate a memory descriptor
      11h "Z_DescFree" release a memory descriptor
      12h "Z_DescGet" get details on memory descriptor
      13h "Z_DescSet" set a memory descriptor
      14h "Z_MemDescAlloc" allocate a memory block and its descriptor
      15h "Z_MemDescFree" release a memory block and its descriptor
      16h "Z_MemDescSizeGet" get size of memory block
      17h "Z_MemDescResize" resize a memory block
      18h "Z_PageLock" lock a page, with existing contents
      19h "Z_PageUnlock" unlock a page
      1Ah "Z_PageLockAny" lock a page, contents undefined
      1Bh "Z_PageUnlockReuse" unlock page, reuse phsyical memory
      1Ch "Z_PageLockNone" lock a page, no physical memory assigned
      1Dh "Z_PageUnlockNone" unlock a page, don't reuse physical memory
      1Eh "F_CallRealRaw" call real mode (SS:ESP supplied)
      1Fh "F_IntRealRaw" perform real-mode interrupt (SS:ESP supplied)
      20h "F_CallReal" call real mode with RETF frame
      21h "F_IntReal" perform real-mode interrupt
      22h "F_PagedCallReal" paged real-mode call
      23h "F_PagedIntReal" paged real-mode interrupt
      24h "F_CallIretReal" call real mode with IRET frame
```

```
      25h "F_CallIretRealRaw" call real mode with IRET (SS:ESP supplied)
      26h "F_CallProt16" call 16-bit protected-mode code
      27h "F_CallProt32" call 32-bit protected-mode code
      28h "F_IAddPage" add kernel instance data
      29h "Z_PageDomLock" lock page in specific domain, preserve contents
      2Ah "Z_PatgeDomUnlock" unlock page in specific domain
      2Bh "Z_PageDomLockAny" lock page in specific domain, undef contents
      2Ch "Z_PageDomUnlockReuse" unlock page in spec. domain,reuse ph.mem
      2Dh "Z_PageDomLockNone" lock page in spec. domain, no phys memory
      2Eh "Z_PageDomUnlockNone" unlock page in spec. domain, no phys. mem
      2Fh "Z_GateAlloc" allocate a gate
      30h "Z_GateFree" release a gate
      31h "X_MemFreeGet" get total free memory
      32h "X_MemTopGet" get highest memory address
      33h "X_MemTotalGet" get total memory size
      34h "F_DescRead" ???
      35h "F_DescWrite" ???
      36h "F_GetStack" ???
      37h "F_SetStack" ???
Return: CX = status (most subfunctions)
  (E)AX and/or (E)BX contain return values, depending on function
Notes:  called by DPMS.EXE and EMM386.EXE
  this API is only available if AX=12FFh/BX=0EDCh returns successfully;
    because the request is handled on the initial trap to the memory
    manager caused by INT instructions, this API must be invoked with
    an actual INT 2F instruction instead of some simulation such as a
    far call to the address in the interrupt vector table
SeeAlso: AX=12FFh/BX=0EDCh,AX=2780h/CL=01h,AX=2780h/CL=02h,AX=2780h/CL=04h
--------m-2F2780CL04-------------------------
INT 2F U - Novell DOS 7 - EMM386.EXE - MULTITASKING API - DOMAINS
  AX = 2780h
  CL = 04h (module: Domains)
  CH = subfunction
      00h "F_DAlloc" create new domain
      01h "F_DFree" delete a domain
      02h "X_DomHandleGetMy" get current domain handle
      03h "F_DSet" switch to another domain
      04h "Z_HandlerSWInt" install software-interrupt handler
      05h "Z_HandlerPageFault" install page-fault handler
      06h "Z_HandlerIOEx" install I/O exception handler
      07h "Z_HandlerGenEx" install general exception handler
```

```
08h "Z_HandlerHWInt" install hardware interrupt handler
09h "Z_IOBitmapGet" get current domain's I/O bitmap entry
0Ah "Z_IOBitmapSet" set current domain's I/O bitmap entry
0Bh "Z_IOBitmapDomGet" get domain's I/O bitmap entry
0Ch "Z_IOBitmapDomSet" set domain's I/O bitmap entry
0Dh "Z_DomMemRead" read memory in another domain
0Eh "Z_DomMemWrite" write memory in another domain
0Fh "Z_PtblDomGet" read domain's page table
10h "Z_PtblDomSet" write domain's page table
11h "Z_InstanceSet" register instanced memory
12h "X_DomNProcessesGet" get number of processes in domain
13h "X_DomSuspend" suspend a domain
14h "Z_DomUnsuspend" resume execution of a domain
15h "Z_DomFork" make a copy of the current domain
16h "Z_DomTerm" terminate domain and all processes in it
17h "Z_HandlerUnlink" remove a handler
18h "Z_HandlerHWIntDflt" install default hardware interrupt handler
19h "Z_HandlerVHWInt" install virtual hardware interrupt handler
1Ah "Z_HandlerVHWIntDflt" install default virtual hware int handler
1Bh "Z_HandlerSwapIn" install swap-in handler
1Ch "Z_HandlerSwapOut" install swap-out handler
1Dh "Z_EndOfInterrupt" signal EOI
1Eh ???
1Fh "Z_HandlerPCreate" install process-creation handler
20h "Z_HandlerPTerm" install process-termination handler
21h "Z_DomRootProcessGet" get domain's root process' handle
22h "F_DForeground" domain has just switched to foreground
23h "F_DBackground" domain has just switched to background
24h "F_MapHMA" map domain's HMA
25h "F_AddInstData" add DOS instance data
26h "X_DomMemFreeGet" get free memory in a domain
27h "X_DomMemUsedGet" get memory used by current domain
28h "Z_DomMemMaxGet" get per-domain memory limit
29h "Z_DomMemMaxSet" set per-domain memory limit
2Ah "F_ReflectInt21" protected-mode INT 21 support
2Bh "Z_DomHandleGet" get domain for a process
2Ch ???
2Dh ???
2Eh ???
2Fh ???
30h ???
```

```
      31h ???
      32h ???
Return: CX = status (most subfunctions)
  (E)AX and/or (E)BX contain return values, depending on function
Notes:  called by DPMS.EXE and EMM386.EXE
  this API is only available if AX=12FFh/BX=0EDCh returns successfully;
    because the request is handled on the initial trap to the memory
    manager caused by INT instructions, this API must be invoked with
    an actual INT 2F instruction instead of some simulation such as a
    far call to the address in the interrupt vector table
SeeAlso: AX=12FFh/BX=0EDCh,AX=2780h/CL=01h,AX=2780h/CL=02h,AX=2780h/CL=03h
--------m-2F2780CL05-------------------------
INT 2F U - Novell DOS 7 - EMM386.EXE - MULTITASKING API - VIRTUAL MACHINES
  AX = 2780h
  CL = 05h (module: VM)
  CH = function
      00h "F_VMProtInit" internal protected-mode initialization
      01h "Z_TMLoad" load task manager
      02h "Z_TMUnload" unload task manager
      03h "Z_VMBackSet" send virtual machine to background
      04h "Z_VMForeSet" send virtual machine to foreground
      05h "Z_VMSaveEnable" enable saving virtual machine
      06h "Z-VMSaveDisable" disable saving virtual machine
      07h "Z_TMInit" intialize task manager
      08h "Z_TMHotKeyGet" get task manager hotkey
      09h "Z_TMHitKeyEnable" restart scanning for task manager key
      0Ah "Z_TMHotKeyDisable" stop scanning for task manager key
      0Bh "X_ForeGet" get current foreground domain
      0Ch "F_DMAHandlerEnable" (re)enable DMA address translation
      0Dh "F_DMAHandlerDisable" disable DMA address translation
      0Eh "Z_SerialTimeoutGet" set serial port timeout
      0Fh "Z_ParallelTimeoutSet" set parallel port timeout
      10h "F_VCPIEntry" domain is entering VCPI mode
      11h "F_VCPIExit" domain is leaving VCPI mode
      12h "X_MsgGlobalEnter" enter Global Message Mode
      13h "X_MsgGlobalDisplay" display global message
      14h "X_MsgGlobalExit" leave Global Message Mode
      15h "X_MsgDomEnter" enter Domain Message Mode
      16h "X_MsgDomDisplay" display domain-specific message
      17h "X_MsgDomExit" leave Domain Message Mode
      18h "X_MsgFatalDisplay" display fatal error message
```

```
      19h "Z_SerialBaseSet" set serial port address
      1Ah "Z_ParallelBaseSet" set parallel port address
      1Bh "Z_SerialIRQSet" set serial port IRQ
      1Ch "Z_ParallelIRQSet" set parallel port IRQ
      1Dh "F_ResetVideo" emergency video system reset
      1Eh "F_SetMouseIRQ" set which IRQ is used by mouse
      1Fh "F_CheckNotIdle" check whether system is idle
      20h "F_GetMouseInfo"
      21h "Z_SerialOwnerGet" get serial port owner
      22h "Z_SerialTimeoutGet" get serial port timeout
      23h "Z_ParallelOwnerGet" get parallel port owner
      24h "Z_ParallelTimeoutGet" get parallel port timeout
  ???
Return: ???
Notes:  called by DPMS.EXE and EMM386.EXE
  the handler for this function may be set by one of the subfunctions
    of AX=2780h/CL=01h; the default handler returns AX=BX=FFFFh and
    CX=0001h (see #02739)
  this API is only available if AX=12FFh/BX=0EDCh returns successfully;
    because the request is handled on the initial trap to the memory
    manager caused by INT instructions, this API must be invoked with
    an actual INT 2F instruction instead of some simulation such as a
    far call to the address in the interrupt vector table
SeeAlso: AX=12FFh/BX=0EDCh,AX=2780h/CL=01h,AX=2780h/CL=02h,AX=2780h/CL=03h
--------m-2F2780CL06-------------------------
INT 2F U - Novell DOS 7 - EMM386.EXE - MULTITASKING API - DPMI SERVICES
  AX = 2780h
  CL = 06h
  CH = function
      00h "F_InitDPMI" used for internal initialization
      01h "F_DescAllocLDT" allocate LDT descriptor
      02h "F_DescFreeLDT" free LDT descriptor
      03h "F_DescGetLDT" get details on LDT descriptor
      04h "F_DescSetLDT" set up LDT descriptor
      05h "F_DescAllocInt21" allocate descriptor for PM INT 21
      06h "F_WhereIsDPMI" get address of DPMI global data
Return: ???
Notes:  called by DPMS.EXE and EMM386.EXE
  the handler for this function may be set by one of the subfunctions
    of AX=2780h/CL=01h; the default handler returns AX=BX=FFFFh and
    CX=0001h (see #02739)
```

```
  this API is only available if AX=12FFh/BX=0EDCh returns successfully;
    because the request is handled on the initial trap to the memory
    manager caused by INT instructions, this API must be invoked with
    an actual INT 2F instruction instead of some simulation such as a
    far call to the address in the interrupt vector table
SeeAlso: AX=12FFh/BX=0EDCh,AX=2780h/CL=01h,AX=2780h/CL=02h,AX=2780h/CL=03h
--------m-2F2780-----------------------------
INT 2F U - Novell DOS 7 - EMM386.EXE - MULTITASKING API - AVAILABLE FOR APPS
  AX = 2780h
  CL = function (07h-0Fh)
  ???
Return: ???
Notes:  called by DPMS.EXE and EMM386.EXE
  the handlers for each of these functions may be set individually by
    one of the subfunctions of AX=2780h/CL=01h; the default handlers
    return with all registers unchanged
  this API is only available if AX=12FFh/BX=0EDCh returns successfully;
    because the request is handled on the initial trap to the memory
    manager caused by INT instructions, this API must be invoked with
    an actual INT 2F instruction instead of some simulation such as a
    far call to the address in the interrupt vector table
SeeAlso: AX=12FFh/BX=0EDCh,AX=2780h/CL=01h,AX=2780h/CL=02h,AX=2780h/CL=03h
--------T-2F2781-----------------------------
INT 2F U - Novell DOS 7 TaskMGR - BEGIN CRITICAL SECTION???
  AX = 2781h
Return: ???
SeeAlso: AX=2782h
--------T-2F2782-----------------------------
INT 2F U - Novell DOS 7 TaskMGR - END CRITICAL SECTION???
  AX = 2782h
Return: ???
SeeAlso: AX=2781h
--------m-2F2783-----------------------------
INT 2F U - Novell DOS 7 - EMM386.EXE - GET ???
  AX = 2783h
Return: AX = ???
  BX = ???
--------T-2F278F-----------------------------
INT 2F U - Novell DOS 7 TaskMGR - ??? API
  AX = 278Fh
  as for INT 2F/AX=2780h
```

```
Return: as for INT 2F/AX=2780h
Note: Novell DOS 7 TaskMGR passes this call through to INT 2F/AX=2780h
    without changing any other registers
SeeAlso: AX=2780h/CL=01h,AX=2782h
--------F-2F2A------------------------------
INT 2F - Gammafax DOS Dispatcher INTERFACE
  AH = 2Ah
Note: details not available at this time
SeeAlso: AX=8000h"FaxBIOS",AX=C000h/BX=444Bh,AX=CB00h,AX=CBDDh,INT 66"BitFax"
--------V-2F2E00----------------------------
INT 2F U - Novell DOS 7 - GRAFTABL - INSTALLATION CHECK
  AX = 2E00h
Return: AH = FFh if installed
Note: this installation check does not follow the usual format of setting
    AL to FFh
SeeAlso: AX=2300h,AH=2Eh"GRAFTABL"
--------V-2F2E------------------------------
INT 2F U - Novell DOS 7 - GRAFTABL - GET FONT TABLE
  AH = 2Eh
  AL nonzero
Return: AH = FFh if installed
      ES:BX -> graphics data (8 bytes per character from 80h to FFh)
SeeAlso: AX=2E00h,AH=23h"GRAFTABL"
--------t-2F3900----------------------------
INT 2F - Kingswood TSR INTERFACE - COMPATIBILITY MODE
  AX = 3900h
Return: AL = status
      00h not installed
      FFh one or more TSRs using this interface is installed
    DX may be destroyed
Note: this function is provided to that the multiplex number will appear used
    to other programs
SeeAlso: AH=39h/BL=00h
--------t-2F39--BL00------------------------
INT 2F - Kingswood TSR INTERFACE - INSTALLATION CHECK
  AH = 39h
  BL = 00h
  AL = TSR ID number (01h-FFh, currently only 01h-1Bh used) (see #02742)
Return: AL = status
      00h not installed
      FFh installed
```

```
    DX = segment address of resident module
Note: All of Kingswood Software's TSRs use this interface.  Usually the
    resident module is installed by allocating a block of upper memory,
    setting its owner ID to 000Ah (used by DOS), and filling the MCB name
    field with the TSR's name.
SeeAlso: #02743,AX=3900h,AH=39h/BL=01h

(Table 02742)
Values for Kingswood TSR ID number:
 01h  TSR Windows
 02h  NOBUSY
 03h  CD STACK
 04h  DISK WATCH
 05h  PUSHBP
 06h  ALIAS
 07h  KEYMACRO
 08h  SLOWDOWN
 09h  ANSIGRAB
 0Ah  TEE
 0Bh  FASTMOUS
 0Ch  EXTWILD
 0Dh  BREAKOUT
 0Eh  STOPDISK
 0Fh  MEMINIT
 10h  JANUSEXT
 11h  CAPS
 12h  ANSI
 13h  TRAPPER
 14h  EATMEM
 15h  WPJOKE
 16h  SHOWDOS
 17h  LOGINTS
 18h  BLANKVGA
 19h  SWAPEXEC
 1Ah  SHELL
 1Bh  TSRGAMES

Format of Kingswood TSR modules:
Offset  Size  Description (Table 02743)
 00h  4 BYTEs signature "FTSR"
 04h  WORD  segment address of this module (used to check validity)
```

```
 06h  WORD  number of words to skip (usually 0000h if no PSP present)
 08h  N WORDs module-defined data that must be at a fixed segment offset
    (usually only a PSP if file access is required)
     5N BYTEs interrupt list (see #02744)
  BYTE  FFh terminator

Format of Kingswood TSR interrupt list entry:
Offset  Size  Description (Table 02744)
 00h  BYTE  interrupt number (00h-FEh)
 01h  WORD  offset within segment of DWORD pointer to previous interrupt
 03h  WORD  offset within segment of begin of interrupt handler code
--------t-2F39--BL01-------------------------
INT 2F - Kingswood TSR INTERFACE - REMOVAL CHECK
  AH = 39h
  BL = 01h
  AL = TSR ID number (01h-FFh) (see #02742)
Return: AL = status
      00h not ready to be removed
      FFh resident module may be removed by deassigning the interrupts
      hooked by the TSR and deallocating the TSR's memory block
  AH,BX,CX,DX,ES may be destroyed
SeeAlso: AX=3900h,AH=39h/BL=00h
--------t-2F39-------------------------------
INT 2F - Kingswood TSR INTERFACE - APPLICATION-SPECIFIC FUNCTION CALLS
  AH = 39h
  BL = function number (02h-FFh)
  AL = TSR ID number (01h-FFh) (see #02742)
  CX,DX,SI,DI,DS,ES may contain parameters
  BH reserved for use by the function dispatcher
Return: as appropriate for the called function
SeeAlso: AX=3900h,AH=39h/BL=00h,AX=3901h/BL=02h
--------r-2F3901BL02-------------------------
INT 2F - Kingswood TSR Windows - OPEN WINDOW
  AX = 3901h
  BL = 02h
Return: AX = error code (0000h if successful)
  SI,DI,DS,ES preserved
Notes:  opens the next TSR window on top of any others.  Only three
    TSR windows can be opened at any one time.  The three windows
    are all 40x11 characters, partly overlapping.
SeeAlso: AH=39h/BL=00h,AX=3901h/BL=03h,AX=3901h/BL=05h,AX=3901h/BL=06h
```

```
--------r-2F3901BL03-------------------------
INT 2F - Kingswood TSR Windows - HIDE WINDOWS
  AX = 3901h
  BL = 03h
Return: AX = error code (0000h if successful)
  SI,DI,DS,ES preserved
Notes:  Hide any visible TSR windows from view.
SeeAlso: AH=39h/BL=00h,AX=3901h/BL=02h,AX=3901h/BL=05h
--------r-2F3901BL04-------------------------
INT 2F - Kingswood TSR Windows - SHOW WINDOWS
  AX = 3901h
  BL = 04h
Return: AX = error code (0000h if successful)
  SI,DI,DS,ES preserved
Notes:  Re-display all TSR windows after a HIDE WINDOWS call.
SeeAlso: AH=39h/BL=00h,AX=3901h/BL=02h,AX=3901h/BL=03h
--------r-2F3901BL05-------------------------
INT 2F - Kingswood TSR Windows - CLOSE WINDOW
  AX = 3901h
  BL = 05h
Return: AX = error code (0000h if successful)
  SI,DI,DS,ES preserved
Notes:  Close the last opened TSR window.
SeeAlso: AH=39h/BL=00h,AX=3901h/BL=02h
--------r-2F3901BL06-------------------------
INT 2F - Kingswood TSR Windows - SET WINDOW TITLE
  AX = 3901h
  BL = 06h
  DS:SI -> title string
Return: AX = error code (0000h if successful)
  SI,DI,DS,ES preserved
SeeAlso: AH=39h/BL=00h,AX=3901h/BL=02h
--------r-2F3901BL07-------------------------
INT 2F - Kingswood TSR Windows - POSITION CURSOR
  AX = 3901h
  BL = 07h
  CH = Y coordinate (0-10)
  CL = X coordinate (0-39)
Return: AX = error code (0000h if successful)
  SI,DI,DS,ES preserved
Note: the hardware cursor is always disabled when a TSR window is opened;
```

```
    this call only sets a text position
SeeAlso: AH=39h/BL=00h,AX=3901h/BL=08h,AX=3901h/BL=09h
--------r-2F3901BL08-------------------------
INT 2F - Kingswood TSR Windows - DISPLAY STRING
  AX = 3901h
  BL = 08h
  DS:SI -> string
Return: AX = error code (0000h if successful)
  SI,DI,DS,ES preserved
Notes:  The text is not clipped.
  This routine understands Tab, NewLine and Carriage Return
SeeAlso: AH=39h/BL=00h,AX=3901h/BL=07h
--------r-2F3901BL09-------------------------
INT 2F - Kingswood TSR Windows - SCROLL WINDOW
  AX = 3901h
  BL = 09h
  CL = scroll direction: 01h up, FFh down, 00h clear window
Return: AX = error code (0000h if successful)
  SI,DI,DS,ES preserved
SeeAlso: AH=39h/BL=00h,AX=3901h/BL=07h
--------r-2F3901BL0A-------------------------
INT 2F - Kingswood TSR Windows - SOUND BEEPER
  AX = 3901h
  BL = 0Ah
  DX = sound divisor, or 0 for silence.
       (divide 1843200 by required frequency to get value for DX)
  CL = sound length in 18.2 Hz clock ticks
Return: AX = error code (0000h if successful)
  SI,DI,DS,ES preserved
SeeAlso: AH=39h/BL=00h
--------r-2F3901BL0B-------------------------
INT 2F - Kingswood TSR Windows - ADD OR REMOVE USER
  AX = 3901h
  BL = 0Bh
  CL = number of users increment: +1 if adding a new user
          -1 if removing a user
Return: AX = error code (0000h if successful)
  SI,DI,DS,ES preserved
Note: the TSR windows resident module may only be removed when the internal
    user count is zero
SeeAlso: AH=39h/BL=00h,AX=3901h/BL=02h
```

```
--------V-2F3912BL03-------------------------
INT 2F - Kingswood ANSI display driver - SET COMPATIBILITY MODE
  AX = 3912h
  BL = 03h
  CL = new mode (00h fast, FFh BIOS)
Return: AL = old compatibility mode
  SI,DI,DS,ES preserved
SeeAlso: AX=3900h,AX=3912h/BL=04h
--------V-2F3912BL04-------------------------
INT 2F - Kingswood ANSI display driver - SET FLAGS
  AX = 3912h
  BL = 04h
  CL = new flags (see #02745)
Return: AL = old flags
  SI,DI,DS,ES preserved
SeeAlso: AX=3900h,AX=3912h/BL=03h

Bitfields for Kingswood ANSI flags:
Bit(s)  Description (Table 02745)
 0  do not wrap at end of line
 1  wait for beeps to end before displaying next character
 2  do not use graphics cursor
--------W-2F4000-----------------------------
INT 2F - Windows 3+ (OS/2 2.x???) - GET VIRTUAL DEVICE DRIVER (VDD) CAPABILTIES
  AX = 4000h
Return: AL = video virtualization (see #02746)
Notes:  This function is used by display drivers to find out what capabilities
    exist for the VDD driver and also trigger the VDD driver to call
    functions 4005h and 4006h (and 4001h/4002h under OS/2?).  This
    function also gives the Video Driver hardware access to the video
    registers.
  Once the background/foreground callouts have been activated by a
    call to this function, the application *must* handle those callouts
    and save/restore the video memory itself.

(Table 02746)
Values for Windows video virtualization:
 01h  does not virtualize video access
 02h  virtualizes the video when in text mode
 03h  virtualizes the video when in text mode or single plane graphics modes
 04h  virtualizes the video when in text mode, single plane graphics modes,
```

```
    and VGA multiplane modes
 FFh  virtualizes the video fully
--------O-2F4001-----------------------------
INT 2F C - OS/2 compatibility box - SWITCHING DOS TO BACKGROUND
  AX = 4001h
Note: called by OS/2 when the DOS box is about to be placed in the background
    and the video driver should save any necessary state
SeeAlso: AX=4002h,AX=4005h
--------O-2F4002-----------------------------
INT 2F C - OS/2 compatibility box - SWITCHING DOS TO FOREGROUND
  AX = 4002h
Note: called by OS/2 when the DOS box is about to be placed in the foreground
    and the video driver should restore the previously-saved state
SeeAlso: AX=4001h,AX=4006h
--------W-2F4003-----------------------------
INT 2F - Windows 3.x - ENTERING VIDEO DRIVER CRITICAL SECTION
  AX = 4003h
Note: This critical section must be exited within 1 second.
SeeAlso: AX=4004h
--------W-2F4004-----------------------------
INT 2F - Windows 3.x - EXITING VIDEO DRIVER CRITICAL SECTION
  AX = 4004h
SeeAlso: AX=4003h
--------W-2F4005-----------------------------
INT 2F C - Windows 3.x - SWITCHING DOS TO BACKGROUND
  AX = 4005h
Notes:  called by Windows when the DOS box is about to be placed in the
    background and the video driver should save any necessary state
    information (this may be called only in Standard mode)
  this callout is not made unless the application has first called
    AX=4000h
SeeAlso: AX=4001h,AX=4006h
--------W-2F4006-----------------------------
INT 2F C - Windows 3.x - SWITCHING DOS TO FOREGROUND
  AX = 4006h
Notes:  called by Windows when the DOS box is about to be placed in the
    foreground and the video driver should restore any necessary state
    information (this may be called only in Standard mode)
  this callout is not made unless the application has first called
    AX=4000h
SeeAlso: AX=4002h,AX=4005h
```

```
--------W-2F4007-----------------------------
INT 2F - Windows 3.x - ENABLE VDD TRAPPING OF VIDEO REGISTERS
  AX = 4007h
Note: used by Windows Standard mode
--------O-2F4010-----------------------------
INT 2F - OS/2 v2.0+ - INSTALLATION CHECK / GET VERSION
  AX = 4010h
Return: AX = 4010h if OS/2 not installed
  AX = 0000h for OS/2 Warp 3.0
  BX = OS/2 version if installed
Note: OS/2 Warp 3.0
SeeAlso: INT 21/AH=30h,INT 21/AX=3306h
--------O-2F4011-----------------------------
INT 2F - OS/2 - GET VDD API ENTRY POINT
  AX = 4011h
  DS:(E)SI -> ASCIZ name of VDD registered with VDHRegisterAPI
Return: ES:DI -> breakpoint address to call for VDD API, or 0000h:0000h
Note: this function may be invoked from either V86 or protected mode, and
    will return the appropriate address to call for invoking the VDD
    in that mode
SeeAlso: AX=1684h"DEVICE API"
--------V-2F4021-----------------------------
INT 2F UC - Diamond Stealth64 Video - STLTH64.VXD - ???
  AX = 4021h
  ???
Return: ???
SeeAlso: AX=4022h,AX=4023h
--------V-2F4022-----------------------------
INT 2F UC - Diamond Stealth64 Video - STLTH64.VXD - ???
  AX = 4022h
  ???
Return: ???
SeeAlso: AX=4021h,AX=4023h
--------V-2F4023-----------------------------
INT 2F UC - Diamond Stealth64 Video - STLTH64.VXD - ???
  AX = 4023h
  ???
Return: ???
SeeAlso: AX=4021h,AX=4022h
--------V-2F4027-----------------------------
INT 2F UC - Diamond Stealth64 Video - DMSSTL.DRV - ???
```

```
  AX = 4027h
  ???
Return: ???
Notes:  called when ???
  a protected-mode handler for this function may be installed with the
    function RFV_HOOKINT2FHANDLER
SeeAlso: AX=4021h,AX=4022h
--------E-2F4040-----------------------------
INT 2F - PharLap 286|DOS-Extender Lite v2.5 - ???
  AX = 4040h
Return: BX:CX -> ???
--------N-2F4100-----------------------------
INT 2F - DOS Enhanced LAN Manager 2.0+ MINIPOP/NETPOPUP - INSTALLATION CHECK
  AX = 4100h
Return: CF clear if successful
      AL = FFh
  CF set on error
      AX = ???
Notes:  MINIPOP and NETPOPUP provide a network message popup service
  LAN Manager enhanced mode adds features beyond the standard redirector
    file/printer services
SeeAlso: AX=118Ah,AX=4103h,AX=4104h,AH=42h,AH=4Bh
--------N-2F4103-----------------------------
INT 2F - DOS Enhanced LAN Manager 2.0+ MINIPOP/NETPOPUP - ???
  AX = 4103h
Return: ???
SeeAlso: AX=4100h,AX=4104h
--------N-2F4104-----------------------------
INT 2F - DOS Enhanced LAN Manager 2.0+ MINIPOP/NETPOPUP - ???
  AX = 4104h
Return: ???
SeeAlso: AX=4100h,AX=4103h
--------N-2F42-------------------------------
INT 2F - LAN Manager 2.0 DOS Enhanced MSRV.EXE - MESSENGER SERVICE
  AH = 42h
  ???
Return: ???
Note: LAN Manager enhanced mode adds features beyond the standard redirector
    file/printer services
SeeAlso: AX=118Ah,AX=4100h,AH=4Bh
--------m-2F4300-----------------------------
```

```
INT 2F - EXTENDED MEMORY SPECIFICATION (XMS) v2+ - INSTALLATION CHECK
  AX = 4300h
Return: AL = 80h XMS driver installed
  AL <> 80h no driver
Notes:  XMS gives access to extended memory and noncontiguous/nonEMS memory
    above 640K
  this installation check DOES NOT follow the format used by other
    software
SeeAlso: AX=4310h,AX=1687h,INT 67/AH=40h,@xxxxh:xxxxh"PMM"
Index:  installation check;XMS version 2+
--------m-2F4308-----------------------------
INT 2F U - HIMEM.SYS v2.77+ - GET A20 HANDLER NUMBER
  AX = 4308h
Return: AL = 43h if supported
      BL = A20 handler number (value of /MACHINE:nn switch)
      BH = AT A20 switch time (00h medium, 01h fast, 02h slow)
Note: if the A20 handler number returned in BL is 00h, an external handler
    is being used (see AX=4330h)
SeeAlso: AX=4309h,AX=4330h
--------m-2F4309-----------------------------
INT 2F U - HIMEM.SYS v3.09+ - GET XMS HANDLE TABLE
  AX = 4309h
Return: AL = 43h if function supported
      ES:BX -> XMS handle table (see #02747)
Note: HIMEM.SYS v3.09 is part of MS-DOS 6.0.
SeeAlso: AX=4308h

Format of XMS handle table:
Offset  Size  Description (Table 02747)
 00h  BYTE  ??? (01h in HIMEM.SYS v3.09)
 01h  BYTE  size of one handle descriptor
 02h  WORD  number of handles (default = 20h)
 04h  DWORD pointer to XMS handle array (see #02748)
SeeAlso: #02777

Format of XMS handle descriptor [array]:
Offset  Size  Description (Table 02748)
 00h  BYTE  flag
    01h=free, 02h=used, 04h=in pool but not associated with any EMB
 01h  BYTE  lock count (00h=unlocked)
 02h  DWORD address of XMS block in KB (shift left by 10 for abs. address)
```

```
 06h  DWORD size of XMS block in KB
--------m-2F4310-----------------------------
INT 2F - EXTENDED MEMORY SPECIFICATION (XMS) v2+ - GET DRIVER ADDRESS
  AX = 4310h
Return: ES:BX -> driver entry point (see #02749,#02750,#02753,#02760,#02769,#02774)
Notes:  HIMEM.SYS v2.77 chains to previous handler if AH is not 00h or 10h
  HIMEM.SYS requires at least 256 bytes free stack space when calling
    the driver entry point
SeeAlso: AX=4300h,AX=4310h"Cloaking",AX=4310h"Netroom",AX=4310h"XMZ"

Format of XMS driver entry point:
Offset  Size  Description (Table 02749)
 00h  5 BYTEs jump to actual handler
    either short jump (EBh XXh) followed by three NOPs or
      far jump (EAh XXXX:XXXX) to a program which has hooked itself
      into the XMS driver chain
Note: to hook into the XMS driver chain, a program should follow the chain of
    far jumps until it reaches the short jump of the driver at the end
    of the chain; this short jump is to be replaced with a far jump to
    the new handler's entry point, which should contain a short jump
    followed by three NOPs.  The new handler must return to the address
    pointed at by the short jump which was overwritten.  Using this
    method, the new handler becomes the first to see every XMS request.

(Table 02750)
Call the XMS driver "Get XMS version number" function with:
  AH = 00h
Return: AX = XMS version (in BCD, AH=major, AL=minor)
  BX = internal revision number (in BCD for HIMEM.SYS)
  DX = High Memory Area (HMA) state
       0001h HMA (1M to 1M + 64K) exists
       0000h HMA does not exist
SeeAlso: #02751,#02752,#02757,#02758,#02764

(Table 02751)
Call the XMS driver "Request High Memory Area" function with:
  AH = 01h
  DX = memory in bytes (for TSR or device drivers)
       FFFFh if application program
Return: AX = status
       0001h success
```

```
      0000h failure
    BL = error code (80h,81h,90h,91h,92h) (see #02775)
Note: HIMEM.SYS will fail function 01h with error code 91h if AL=40h and
    DX=KB free extended memory returned by last call of function 08h
SeeAlso: #02752,#02784

(Table 02752)
Call the XMS driver "Release High Memory Area" function with:
  AH = 02h
Return: AX = status
      0001h success
      0000h failure
    BL = error code (80h,81h,90h,93h) (see #02775)
SeeAlso: #02751

(Table 02753)
Call the XMS driver "Global enable A20, for using the HMA" function with:
  AH = 03h
Return: AX = status
      0001h success
      0000h failure
    BL = error code (80h,81h,82h) (see #02775)
SeeAlso: #02754,#02755,MSR 00001000h

(Table 02754)
Call the XMS driver "Global disable A20" function with:
  AH = 04h
Return: AX = status
      0001h success
      0000h failure
    BL = error code (80h,81h,82h,94h) (see #02775)
SeeAlso: #02753,#02756,MSR 00001000h

(Table 02755)
Call the XMS driver "Local enable A20" function with:
  AH = 05h
Return: AX = status
      0001h success
      0000h failure
    BL = error code (80h,81h,82h) (see #02775)
Note: this function is used for direct access to extended memory
```

```
SeeAlso: #02753,#02756

(Table 02756)
Call the XMS driver "Local disable A20" function with:
  AH = 06h
Return: AX = status
      0001h success
      0000h failure
    BL = error code (80h,81h,82h,94h) (see #02775)
SeeAlso: #02754,#02755

(Table 02757)
Call the XMS driver "Query A20 state" function with:
  AH = 07h
Return: AX = status
      0001h enabled
      0000h disabled
    BL = error code (00h,80h,81h) (see #02775)
SeeAlso: #02750,#02758

(Table 02758)
Call the XMS driver "Query free extended memory" function with:
  AH = 08h
  BL = 00h (some implementations leave BL unchanged on success)
Return: AX = size of largest extended memory block in KB
  DX = total extended memory in KB
  BL = error code (00h,80h,81h,A0h) (see #02775)
Note: this function does not include the HMA in the returned memory sizes
SeeAlso: #02750,#02757,#02759,#02771

(Table 02759)
Call the XMS driver "Allocate extended memory block" function with:
  AH = 09h
  DX = Kbytes needed
Return: AX = status
      0001h success
    DX = handle for memory block
      0000h failure
    BL = error code (80h,81h,A0h) (see #02775)
SeeAlso: #02758,#02761,#02764,#02765,#02766,#02772
```

```
(Table 02760)
Call the XMS driver "Free extended memory block" function with:
  AH = 0Ah
  DX = handle of block to free
Return: AX = status
      0001h success
      0000h failure
    BL = error code (80h,81h,A2h,ABh) (see #02775)
SeeAlso: #02759,#02772

(Table 02761)
Call the XMS driver "Move extended memory block" function with:
  AH = 0Bh
  DS:SI -> EMM structure (see #02776)
Return: AX = status
      0001h success
      0000h failure
    BL = error code (80h-82h,A3h-A9h) (see #02775)
Note: if either handle in the EMM structure is 0000h, the corresponding
    offset is considered to be an absolute segment:offset address in
    directly addressable memory
SeeAlso: #02759,#02762

(Table 02762)
Call the XMS driver "Lock extended memory block" function with:
  AH = 0Ch
  DX = handle of block to lock
Return: AX = status
      0001h success
    DX:BX = 32-bit physical address of locked block
      0000h failure
    BL = error code (80h,81h,A2h,ACh,ADh) (see #02775)
Note: MS Windows 3.x rejects this function for handles allocated after
    Windows started
SeeAlso: #02759,#02761,#02763,#02777

(Table 02763)
Call the XMS driver "Unlock extended memory block" function with:
  AH = 0Dh
  DX = handle of block to unlock
Return: AX = status
```

```
      0001h success
      0000h failure
    BL = error code (80h,81h,A2h,AAh) (see #02775)
SeeAlso: #02762

(Table 02764)
Call the XMS driver "Get handle information" function with:
  AH = 0Eh
  DX = handle for which to get info
Return: AX = status
      0001h success
    BH = block's lock count
    BL = number of free handles left
    DX = block size in KB
      0000h failure
    BL = error code (80h,81h,A2h) (see #02775)
BUG:  MS Windows 3.10 acts as though unallocated handles are in use
Note: MS Windows 3.00 has problems with this call
SeeAlso: #02750,#02759,#02773

(Table 02765)
Call the XMS driver "Reallocate extended memory block" function with:
  AH = 0Fh
  DX = handle of block
  BX = new size of block in KB
Return: AX = status
      0001h success
      0000h failure
    BL = error code (80h,81h,A0h-A2h,ABh) (see #02775)
SeeAlso: #02759,#02768

(Table 02766)
Call the XMS driver "Request upper memory block" function with:
  AH = 10h
  DX = size of block in paragraphs
Return: AX = status
      0001h success
    BX = segment address of UMB
    DX = actual size of block
      0000h failure
    BL = error code (80h,B0h,B1h) (see #02775)
```

```
    DX = largest available block
Notes:  Upper Memory consists of non-EMS memory between 640K and 1024K
  the XMS driver need not implement functions 10h through 12h to be
    considered compliant with the standard
  under DOS 5+, if CONFIG.SYS contains the line DOS=UMB, then no upper
    memory blocks will be available for allocation because all blocks
    have been grabbed by MS-DOS while booting
SeeAlso: #02759,#02767,#02785,INT 21/AH=58h"UMB"

(Table 02767)
Call the XMS driver "Release upper memory block" function with:
  AH = 11h
  DX = segment address of UMB to release
Return: AX = status
      0001h success
      0000h failure
    BL = error code (80h,B2h) (see #02775)
Note: the XMS driver need not implement functions 10h through 12h to be
    considered compliant with the standard
SeeAlso: #02760,#02766,#02768

(Table 02768)
Call the XMS v3.0+ driver "Reallocate upper memory block" function with:
  AH = 12h
  DX = segment address of UMB to resize
  BX = new size of block in paragraphs
Return: AX = status
      0001h success
      0000h failure
    BL = error code (80h,B0h,B2h) (see #02775)
    DX = maximum available size (RM386)
Note: the XMS driver need not implement functions 10h through 12h to be
    considered compliant with the standard
SeeAlso: #02765,#02766,#02767,#02783

(Table 02769)
Call the QEMM v5.11 "???" function with:
  AH = 34h  (QEMM 5.11 only, undocumented)
  ???
Return: ???
SeeAlso: #02770
```

```
(Table 02770)
Call the QEMM v5.11 "???" function with:
  AH = 44h  (QEMM 5.11 only, undocumented)
  ???
Return: ???
SeeAlso: #02769,#02783

(Table 02771)
Call the XMS v3.0 driver "Query free extended memory" function with:
  AH = 88h
Return: EAX = largest block of extended memory, in KB
  BL = status (00h,80h,81h,A0h) (see #02775)
  ECX = physical address of highest byte of memory
      (valid even on error codes 81h and A0h)
  EDX = total Kbytes of extended memory (0 if status A0h)
BUG:  HIMEM v3.03-3.07 crash on an 80286 machine if any of the 8Xh functions
    are called
SeeAlso: #02758,#02772

(Table 02772)
Call the XMS v3.0 driver "Allocate any extended memory" function with:
  AH = 89h
  EDX = Kbytes needed
Return: AX = status
      0001h success
    DX = handle for allocated block (free with AH=0Ah) (see #02760)
      0000h failure
    BL = status (80h,81h,A0h,A1h,A2h) (see #02775)
SeeAlso: #02759,#02771

(Table 02773)
Call the XMS v3.0 driver "Get extended EMB handle information" function with:
  AH = 8Eh
  DX = handle
Return: AX = status
      0001h success
    BH = block's lock count
    CX = number of free handles left
    EDX = block size in KB
      0000h failure
```

```
    BL = status (80h,81h,A2h) (see #02775)
BUG:  MS-DOS 6.0 HIMEM.SYS leaves CX unchanged
SeeAlso: #02764,#02772,#02774

(Table 02774)
Call the XMS v3.0 driver "Reallocate any extended memory block" function with:
  AH = 8Fh
  DX = unlocked memory block handle
  EBX = new size in KB
Return: AX = status
      0001h success
      0000h failure
    BL = status (80h,81h,A0h-A2h,ABh) (see #02775)
BUG:  HIMEM v3.03-3.07 crash on an 80286 machine if any of the 8Xh functions
    are called
SeeAlso: #02765,#02773

(Table 02775)
Values for XMS error code returned in BL:
 00h  successful
 80h  function not implemented
 81h  Vdisk was detected
 82h  an A20 error occurred
 8Eh  a general driver error
 8Fh  unrecoverable driver error
 90h  HMA does not exist or is not managed by XMS provider
 91h  HMA is already in use
 92h  DX is less than the /HMAMIN= parameter
 93h  HMA is not allocated
 94h  A20 line still enabled
 A0h  all extended memory is allocated
 A1h  all available extended memory handles are allocated
 A2h  invalid handle
 A3h  source handle is invalid
 A4h  source offset is invalid
 A5h  destination handle is invalid
 A6h  destination offset is invalid
 A7h  length is invalid
 A8h  move has an invalid overlap
 A9h  parity error occurred
 AAh  block is not locked
```

```
 ABh  block is locked
 ACh  block lock count overflowed
 ADh  lock failed
 B0h  only a smaller UMB is available
 B1h  no UMB's are available
 B2h  UMB segment number is invalid

Format of EMM structure:
Offset  Size  Description (Table 02776)
 00h  DWORD number of bytes to move (must be even)
 04h  WORD  source handle
 06h  DWORD offset into source block
 0Ah  WORD  destination handle
 0Ch  DWORD offset into destination block
Notes:  if source and destination overlap, only forward moves (source base
    less than destination base) are guaranteed to work properly
  if either handle is zero, the corresponding offset is interpreted
    as a real-mode address referring to memory directly addressable
    by the processor

Format of XMS handle info [array]:
Offset  Size  Description (Table 02777)
 00h  BYTE  handle
 01h  BYTE  lock count
 02h  DWORD handle size
 06h  DWORD handle physical address (only valid if lock count nonzero)
SeeAlso: #02747,#02762
--------m-2F4310-----------------------------
INT 2F - Cloaking - REAL-MODE API
  AX = 4310h
Return: ES:BX -> driver entry point (see #02749,#02778,#02779,#02780,#02781)
SeeAlso: AX=4310h"XMS"

(Table 02778)
Call the Cloaking v1.01 "Client Registration" function with:
  AH = 7Eh
  BX = subfunction
      0000h get client registration count
      0001h get client registration structures
    ES:DI -> buffer for registration structures
Return: AX = status
```

```
      0000h failed
      0001h successful
    ---subfunction 00h---
    BX = size of client structure in bytes
    CX = number of clients installed
    ---subfunction 01h---
    ES:DI buffer filled
SeeAlso: #02779,#02781,INT 2C/AX=0033h

(Table 02779)
Call the Cloaking v1.01 "Verify Cloaking Host" function with:
  AH = 7Fh
Return: AX = status
      0000h failed
      0001h (successful) if installed
    BX = version (0101h for v1.01)
    CX = flags
        bit 0: host is VCPI-based
    DS:DX -> ASCIZ Cloaking host signature
        "CLOAKING.EXE"0, followed by a far-call entry point to
          uninstall host (see #02780) in Helix's CLOAKING.EXE
SeeAlso: #02778,#02781
Index:  installation check;Cloaking host|installation check;CLOAKING.EXE

(Table 02780)
Call the CLOAKING.EXE "Uninstall Host" function with:
Return: AX = 4F4Bh ('OK') if successfully uninstalled protected-mode code

(Table 02781)
Call the Cloaking "Start Protected-Mode Client" function with:
  AH = 82h
  DX = XMS handle of locked block containing protected-mode code
  CL = code size (00h 16-bit, else 32-bit)
  ESI, EDI = parameters to pass to protected-mode code
Return: AX = status
      nonzero success
      0000h failed
    BL = error code (A2h,B0h) (see #02775)
Notes:  this function calls a user initialization function at offset 0 in
    the XMS memory block (see #02782)
  supported by Helix's RM386 v6.00 and Helix's CLOAKING.EXE
```

```
SeeAlso: #02778,#02779

(Table 02782)
Values user initialization function is called with:
  EBX = physical address of block's start
  ESI = user data from function 82h call
  EDI = user data from function 82h call
  CS = code selector for XMS block at EBX (16-bit or 32-bit)
  DS = data selector for XMS block, starting at EBX
  ES = selector for V86 memory access to full real-mode 1088K
  GS = selector for full 4G flat address space
  SS:ESP -> stack provided by host
Return: via 32-bit FAR return
Note: the initialization function may call any protected-mode Cloaking
    service; it should store the values of DS, ES, and GS for future
    reference
--------m-2F4310-----------------------------
INT 2F - Helix Netroom RM386 v6.00 - XMS EXTENSIONS
  AX = 4310h
Return: ES:BX -> driver entry point (see #02783,#02784,#02785,#02786)
Notes:  HIMEM.SYS v2.77 chains to previous handler if AH is not 00h or 10h
  HIMEM.SYS requires at least 256 bytes free stack space when calling
    the driver entry point
SeeAlso: AX=4300h,AX=4310h"XMS",AX=4310h"Cloaking"

(Table 02783)
Call the Netroom RM386 v6.00 "Reallocate upper memory block" function with:
  AH = 80h
  DX = segment address of UMB to resize
  BX = new size of block in paragraphs
Return: AX = status
      0001h success
      0000h failure
    BL = error code (80h,B0h,B2h) (see #02775)
    DX = maximum available size
Note: this function is identical to function 12h
SeeAlso: #02768,#02784

(Table 02784)
Call the Netroom RM386 v6.00 "re-enable HMA allocation" function with:
  AH = 81h
```

```
Return: AX = 0001h (success)
SeeAlso: #02751,#02783,#02785

(Table 02785)
Call the Netroom RM386 v6.00 "Create new UMB entry" function with:
  AH = 83h
  BX = segment of high-memory block
  DX = first page of start of block
  CX = number of consecutive pages in block
  DI = start of UMB in block
Return: AX = 0001h (success)
  DI = segment of first high-DOS block
Note: the new UMB is not linked into the high-memory chain
SeeAlso: #02766,#02784,#02786

(Table 02786)
Call the Netroom RM386 v6.00 "Get all XMS handles info" function with:
  AH = 84h
  CX = size of buffer for handle info
  ES:DI -> buffer for handle info (see #02777)
Return: AX = 0001h (success)
  DX = current number of allocated XMS handles
SeeAlso: #02785,#02771
--------m-2F4310-----------------------------
INT 2F - NEC PC-9800 - XMZ - PRIVATE API
  AX = 4310h
Return: ES:BX -> driver entry point (see #02787,#02788)
Program: XMZ is an XMS 2.x-compatible driver for the NEC PC-98 series written
    by ZOBplus Hayami and available at
    ftp:/ftp.tohoku.ac.jp/pub/msdos/Memory/xmz/
SeeAlso: AX=4300h,AX=4310h"XMS"

(Table 02787)
Call XMZ v1.02 "Get HMA Information" function with:
  AH = FFh  (XMZ only)
  AL = 01h
Return: AX = 1 on success
  DX = minimum HMA allocation size (/HMAMIN=)
  BX = actual size of HMA allocation, if in use (i.e. the value in DX
       when XMS function 1 was called)
SeeAlso: #02788
```

```
(Table 02788)
Call XMZ v1.02 "Get EMB Handle Information" function with:
  AH = FFh  (XMZ only)
  AL = 02h
Return: AX = 1 on success
  DX = number of EMB handles configured (/NUMHANDLES=)
  BX = offset in XMZ's segment of the handle table (use segment of
       entry point) (see #02789)
SeeAlso: #02787

Format of XMZ v1.02 EMB Handle structure:
Offset  Size  Description (Table 02789)
 00h  BYTE  flag byte
    04h unused handle slot
    02h in-use handle slot
    01h handle slot that represents a free block
 01h  BYTE  lock count
 02h  WORD  block start address (1K increments)
 04h  WORD  block length (1K increments)
SeeAlso: #02788
--------m-2F4320-----------------------------
INT 2F U - HIMEM.SYS - Mach 20 SUPPORT
  AX = 4320h
  ???
Return: ???
--------m-2F4330-----------------------------
INT 2F CU - HIMEM.SYS v2.77+ - GET EXTERNAL A20 HANDLER ADDRESS
  AX = 4330h
Return: AL = 80h if external A20 handler provided
      ES:BX -> external A20 handler (see #02790)
      CL = A20 detection support
    00h handler is unable to report A20 state
    01h handler supports function 0002h to report A20 state
Note: HIMEM.SYS calls this function to allow an external program to provide
    an A20 handler (i.e. to support a machine not supported by HIMEM
    itself)
SeeAlso: AX=4308h,AX=4310h

(Table 02790)
Call parameters for external A20 handler are:
```

```
  AX = function
       0000h disable A20
       0001h enable A20
       0002h get A20 state
Return: AX = status (functions 0000h and 0001h)
       0000h failure
       0001h successful
  AX = A20 state (function 0002h)
       0000h disabled
       0001h enabled
Note: HIMEM.SYS only calls function 0002h if the returned CL indicated that
    the handler supports the call
----------2F43D6-----------------------------
INT 2F - Multiplex - ???
  AX = 43D6h
Note: Central Point's CPBACKUP v9 calls this function with CX=07FFh and
    DX=80D3h at startup
--------E-2F43E0BX0000-----------------------
INT 2F - DOS Protected Mode Services (DPMS) v1.0 - INSTALLATION CHECK
  AX = 43E0h
  BX = 0000h
  CX = 4450h ('DP')
  DX = 4D53h ('MS')
Return: AX = 0000h if installed
      CF clear
      ES:DI -> server structure (see #02791)
      ES:BX -> registration structure (pre-NWDOS 7 beta spec) (see #02793)
Note: the DPMS 1.0 server included with the original release of Novell DOS
    7.0 supports both the beta and 1.0 specification, setting ES:BX even
    if CX and DX are not as specified on entry (since the beta
    specification did not use those registers).  However, the DPMS 1.1
    server included with the March 1994 update does not support the beta
    specification
SeeAlso: AX=43E1h,AX=43E2h,AX=43E3h,INT 2F/AX=1687h
Index:  signature strings;DPMS

Format of DPMS 1.0 server structure:
Offset  Size  Description (Table 02791)
 00h  4 BYTEs signature string "DPMS"
 04h  2 BYTEs DPMS version (major,minor)
 06h  8 BYTEs blank-padded server OEM name
```

```
 0Eh  2 BYTEs OEM server version (major,minor)
 10h  WORD  DPMS flags (see #02792)
 12h  BYTE  CPU type
    (02h = 286, 03h = 386 or higher, higher values allowed)

Bitfields for DPMS flags:
Bit(s)  Description (Table 02792)
 0  fast processor reset available (286 only)
 1  DPMS server is enabled
 2  memory is remapped
 3-15 reserved (undefined)

Format of beta DPMS registration structure:
Offset  Size  Description (Table 02793)
 00h  DWORD real-mode API entry point (see #02795)
 04h  DWORD 16-bit protected-mode API entry point (see #02795)
 08h  8 BYTEs reserved (0)
 10h  8 BYTEs blank-padded server OEM name
 18h  WORD  flags
    bit 0: fast processor reset available (286 only)
    bits 1-15 reserved (undefined)
 1Ah  2 BYTEs DPMS version (major,minor)
 1Ch  BYTE  CPU type (02h = 286, 03h = 386 or higher)
--------m-2F43E1-----------------------------
INT 2F - DOS Protected Mode Services (DPMS) v1.0 - REGISTER CLIENT
  AX = 43E1h
  CX = required protected-mode stack size in bytes
  ES:DI -> DPMS client interface structure (see #02794)
Return: AX = 0000h if supported
     CF clear
     ES:DI buffer filled with API entry point code from offset 0Ah
Note: the client is allowed to copy the returned API code to any location in
    memory, and need not keep the three code fields together
SeeAlso: AX=43E0h,AX=43E2h,AX=43E3h

Format of DPMS client interface structure:
Offset  Size  Description (Table 02794)
 00h  WORD  0000h (structure version / flags)
 02h  8 BYTEs blank-padded client name
 0Ah  7 BYTEs real/virtual-86 mode API code (see #02795)
 11h  BYTE  space for return instruction
```

```
    set to C3h for near return, CBh for far return
 12h  7 BYTEs 16-bit protected-mode API code (see #02795)
 19h  BYTE  space for return instruction
    set to C3h for near return, CBh for far return
 1Ah  9 BYTEs 32-bit protected-mode API code (see #02795)
 23h  BYTE  space for return instruction
    set to C3h for near return, CBh for far return
Note: the DPMS server fills the return opcode bytes with zeros and DPMS
    requests will thus crash the system unless the application
    explicitly sets them (some early versions set them to C3h by
    default, but one should not rely on that)

(Table 02795)
Call DPMS entry point with:
  AX = 0000h unregister client from server
 ---control transfer functions---
  AX = 0100h call protected-mode procedure
    CX = number of words of stack to copy
    ES:(E)DI -> callup/down register structure (see #02797)
    Return: CF clear if successful
      CF set on error
          AX = error code (see #02796)
  AX = 0101h call real-mode procedure (RETF return)
    CX = number of words of stack to copy
    ES:(E)DI -> callup/down register structure (see #02797)
    Return: CF clear if successful
      CF set on error
          AX = error code (see #02796)
  AX = 0102h call real-mode procedure (IRET return)
    CX = number of words of stack to copy
    ES:(E)DI -> callup/down register structure (see #02797)
    Return: CF clear if successful
      CF set on error
          AX = error code (see #02796)
  AX = 0103h call real-mode interrupt handler
    BL = interrupt number
    CX = number of words of stack to copy
    ES:(E)DI -> callup/down register structure (see #02797)
    Return: CF clear if successful
      CF set on error
          AX = error code (see #02796)
```

```
AX = 0104h register default protected mode procedure
  ES:(E)DI -> default register structure (see #02798)
  Return: CF clear if successful
    CF set on error
        AX = error code (see #02796)
AX = 0105h register default real-mode procedure (RETF return)
  ES:(E)DI -> default register structure (see #02798)
  Return: CF clear if successful
    CF set on error
        AX = error code (see #02796)
  Note: the procedure will be called from 16-bit prot. mode
AX = 0106h register default real-mode procedure (IRET return)
  ES:(E)DI -> default register structure (see #02798)
  Return: CF clear if successful
    CF set on error
        AX = error code (see #02796)
  Note: the procedure will be called from 16-bit prot. mode
AX = 0107h register default real-mode interrupt handler
  BL = interrupt number
  ES:(E)DI -> default register structure (see #02798)
  Return: CF clear if successful
    CF set on error
        AX = error code (see #02796)
  Note: the handler will be called from 16-bit protected mode
AX = 0108h register default real-mode procedure (RETF return)
  ES:(E)DI -> default register structure (see #02798)
  Return: CF clear if successful
    CF set on error
        AX = error code (see #02796)
  Note: the procedure will be called from 32-bit prot. mode
AX = 0109h register default real-mode procedure (IRET return)
  ES:(E)DI -> default register structure (see #02798)
  Return: CF clear if successful
    CF set on error
        AX = error code (see #02796)
  Note: the procedure will be called from 32-bit prot. mode
AX = 010Ah register default real-mode interrupt handler
  BL = interrupt number
  ES:(E)DI -> default register structure (see #02798)
  Return: CF clear if successful
    CF set on error
```

```
          AX = error code (see #02796)
    Note: the handler will be called from 32-bit protected mode
---descriptor management---
 AX = 0200h allocate descriptors
    CX = number of descriptors to allocate
    Return: CF clear if successful
          AX = selector for first descriptor allocated
      CF set on error
          AX = error code (see #02796)
 AX = 0201h free a descriptor
    BX = selector for descriptor
    Return: CF clear if successful
      CF set on error
          AX = error code (see #02796)
 AX = 0202h create alias descriptor
    BX = selector for descriptor to be aliased
    Return: CF clear if successful
          AX = alias descriptor
      CF set on error
          AX = error code (see #02796)
 AX = 0203h build alias to real-mode segment
    BX = descriptor
    CX = real-mode segment
    Return: CF clear if successful
      CF set on error
          AX = error code (see #02796)
 AX = 0204h set descriptor base
    BX = descriptor
    CX:DX = base address
    Return: CF clear if successful
      CF set on error
          AX = error code (see #02796)
 AX = 0205h set descriptor limit
    BX = descriptor
    CX = limit
    Return: CF clear if successful
      CF set on error
          AX = error code (see #02796)
 AX = 0206h set descriptor type/attribute
    BX = descriptor
    CL = type
```

```
    CH = attribute
    Return: CF clear if successful
      CF set on error
          AX = error code (see #02796)
 AX = 0207h get descriptor base
    BX = descriptor
    Return: CF clear if successful
          CX:DX = base address
      CF set on error
          AX = error code (see #02796)
---linear memory functions---
 AX = 0300h get size of largest free block of memory
    Return: CF clear if successful
          BX:CX = size
      CF set on error
          AX = error code (see #02796)
 AX = 0301h allocate block of extended memory
    BX:CX = required size
    Return: CF clear if successful
          BX:CX = base address
          SI:DI = handle
      CF set on error
          AX = error code (see #02796)
 AX = 0302h free block of extended memory
    SI:DI = handle
    Return: CF clear if successful
      CF set on error
          AX = error code (see #02796)
 AX = 0303h map linear memory
    ES:(E)DI -> DDS (see #02799)
    Return: CF clear if successful
          BX:CX = base address
          SI:DI = handle
      CF set on error
          AX = error code (see #02796)
 AX = 0304h unmap linear memory
    SI:DI = handle
    Return: CF clear if successful
      CF set on error
          AX = error code (see #02796)
 AX = 0305h get page table entries
```

```
    ESI = linear address
    (E)CX = count
    ES:(E)DI -> buffer for page table entries
    Return: CF clear if successful
          ES:(E)DI buffer filled
      CF set on error
          AX = error code (see #02796)
  AX = 0306h set page table entries
    EBX = linear memory handle
    ESI = linear address
    (E)CX = count
    ES:(E)DI -> buffer containing page table entries
    Return: CF clear if successful
      CF set on error
          AX = error code (see #02796)
  AX = 0307h get largest mappable block size
    Return: CF clear if successful
          BX:CX = size
      CF set on error
          AX = error code (see #02796)
 ---miscellaneous---
  AX = 0400h relocate segment to extended memory
    ES:SI = base address
    CX = limit
    BL = type
    BH = attribute
    DX = selector or 0000h
    Return: CF clear if successful
          AX = selector
          BX:CX = new base address
          SI:DI = handle
      CF set on error
          AX = error code (see #02796)
Note: the beta DPMS specification, which is still supported by the Novell
    DOS 7.0 DPMS host, only supported functions 0100h-0103h, 0200h-0207h,
    0300h-0304h, and 0400h

(Table 02796)
Values for DPMS error code:
 8000h  general error
 8001h  unsupported function
```

```
 8002h  unable to switch to protected mode
 8004h  no default stack defined
 8005h  unknown client
 8010h  resource unavailable
 8011h  descriptor unavailable
 8012h  linear memory unavailable
 8013h  physical memory unavailable
 8021h  invalid value
 8022h  invalid selector
 8023h  invalid handle
 8025h  invalid linear address

Format of DPMS callup/down register structure:
Offset  Size  Description (Table 02797)
 00h  DWORD EDI
 04h  DWORD ESI
 08h  DWORD EBP
 0Ch  4 BYTEs reserved (0) (ESP, may be used by DPMS server)
 10h  DWORD EBX
 14h  DWORD EDX
 18h  DWORD ECX
 20h  DWORD EAX
 24h  DWORD EIP
 28h  WORD  CS
 2Ah  2 BYTEs reserved (0)
 2Ch  DWORD EFLAGS
 30h  DWORD ESP
 34h  WORD  SS
 36h  2 BYTEs reserved (0)
 38h  WORD  ES
 3Ah  2 BYTEs reserved (0)
 3Ch  WORD  DS
 3Eh  2 BYTEs reserved (0)
 40h  WORD  FS
 42h  2 BYTEs reserved (0)
 44h  WORD  GS
 46h  2 BYTEs reserved (0)

Format of DPMS default register structure:
Offset  Size  Description (Table 02798)
 00h  DWORD EIP
```

```
 04h  WORD  CS
 06h  2 BYTEs reserved (0)
 08h  WORD  number of words to copy from stack to stack
 0Ah  BYTE  (call) 00h
    (ret) nonzero if call could not be made
 0Bh  BYTE  reserved (may be used by some servers)
 0Ch  DWORD ESP
 10h  WORD  SS
 12h  2 BYTEs reserved (0)
 14h  WORD  ES
 16h  2 BYTEs reserved (0)
 18h  WORD  DS
 1Ah  2 BYTEs reserved (0)
 1Ch  WORD  FS
 1Eh  2 BYTEs reserved (0)
 20h  WORD  GS
 22h  2 BYTEs reserved (0)
 24h  9 BYTEs API entry code (filled in by server)

Format of DPMS lock DDS:
Offset  Size  Description (Table 02799)
 00h  DWORD total size in bytes
 04h  DWORD offset
 08h  WORD  segment or selector
 0Ah  WORD  reserved
 0Ch  WORD  maximum number of physical blocks structure has space for
 0Eh  WORD  number of physical blocks listed
 10h  DWORD physical address of first block
 14h  DWORD size in bytes of first block
  ...
--------m-2F43E2-----------------------------
INT 2F - DOS Protected Mode Services (DPMS) v1.0 - ENABLE/DISABLE DPMS
  AX = 43E2h
  BX = new state (0000h disable, 0001h enable)
Return: AX = 0000h if supported
Note: this function should normally be called only by system software
SeeAlso: AX=43E0h,AX=43E1h,AX=43E3h
--------m-2F43E3BX0000----------------------
INT 2F C - DOS Protected Mode Services (DPMS) v1.0 - DPMS STARTUP BROADCAST
  AX = 43E3h
  BX = 0000h
```

```
  CX = 4450h ('DP')
  DX = 4D53h ('MS')
  ES:DI -> DPMS server structure (see #02791)
Return: BX = status
      bit 0 set if server is not allowed to load (e.g. some resident
      program is incompatible)
Note: this callout is made while the potential server is running as a normal
    real-mode DOS program, so there are no BIOS/DOS re-entrancy issues
SeeAlso: AX=43E0h,AX=43E4h
--------m-2F43E4BX0000----------------------
INT 2F C - DOS Protected Mode Services (DPMS) v1.0 - DPMS EXIT BROADCAST
  AX = 43E4h
  BX = 0000h
  CX = 4450h ('DP')
  DX = 4D53h ('MS')
Return: BX = status
      bit 0 set if server is not allowed to unload
Note: servers never unload until all clients unregister
SeeAlso: AX=43E0h,AX=43E3h
--------E-2F44------------------------------
INT 2F U - DOS Extender support???
  AH = 44h
  AL = function (at least 0Bh, 15h, 17h)
  ???
Return: ???
Note: called by Codeview for Windows
SeeAlso: AH=86h
--------G-2F4500----------------------------
INT 2F U - Microsoft Profiler (PROF.COM/VPROD.386) - INSTALLATION CHECK
  AX = 4500h
Return: AL = installation status
      01h if PROF.COM installed
      02h if VPROD.386 installed
SeeAlso: AX=4501h,AX=4502h
--------G-2F4501----------------------------
INT 2F U - Microsoft Profiler (PROF.COM/VPROD.386) - SETUP PROFILER
  AX = 4501h
  BX = CSIPS buffer size in KB (first parameter for ProfSetup)
  CX = output limit in KB (second parameter for ProfSetup)
Note: this call is not supported by PROF.COM
SeeAlso: AX=4502h,AX=4503h
```

```
--------G-2F4502-----------------------------
INT 2F U - Microsoft Profiler (PROF.COM/VPROD.386) - SET SAMPLING RATE
  AX = 4502h
  BL = sampling rate for PROF.COM (0 < BL <= 13)
       (01h = 8192/s, 04h = 1024/s, 08h = 32/s, 0Dh = 1/s)
  CX = sampling rate for VPROD.386
Note: for PROF.COM, this programs the CMOS clock by setting BL+2 as the
    low four bits of CMOS register 0Ah.  The interruption rate is
    1 SHL (15 - BL) per second.
SeeAlso: AX=4501h,AX=4503h
--------G-2F4503-----------------------------
INT 2F U - Microsoft Profiler (PROF.COM/VPROD.386) - START PROFILING
  AX = 4503h
Notes:  Profiling is also turned on by the key combinations
    LeftShift + RightShift + Alt and LeftShift + RightShift + Ctrl
  for PROF.COM, this call programs the CMOS clock by reading register
    0Ch, and setting bit 6 of register 0Bh.  It then makes sure that IRQ8
    is unmasked
SeeAlso: AX=4504h
--------G-2F4504-----------------------------
INT 2F U - Microsoft Profiler (PROF.COM/VPROD.386) - STOP PROFILING
  AX = 4504h
Notes:  profiling is also turned off by the key combination
    LeftShift + RightShift
  for PROF.COM, this programs the CMOS clock by reading register 0Ch
    and clearing bit 6 of register 0Bh.  It then masks IRQ8.
SeeAlso: AX=4503h,AX=4505h,AX=4506h,AX=4507h
--------G-2F4505-----------------------------
INT 2F U - Microsoft Profiler (PROF.COM/VPROD.386) - CLEAR PROFILING DATA
  AX = 4505h
SeeAlso: AX=4503h,AX=4504h,AX=4506h
--------G-2F4506-----------------------------
INT 2F U - Microsoft Profiler (PROF.COM/VPROD.386) - "ProfFlush"
  AX = 4506h
SeeAlso: AX=4505h,AX=4507h
--------G-2F4507-----------------------------
INT 2F U - Microsoft Profiler (PROF.COM/VPROD.386) - "ProfFinish"
  AX = 4507h
Note: this call is essentially a "ProfStop" (AX=4504h) followed by
    "ProfFlush" (AX=4506h)
SeeAlso: AX=4504h,AX=4505h,AX=4506h
```

```
--------G-2F4508-----------------------------
INT 2F U - Microsoft Profiler (PROF.COM/VPROD.386) - ALTERNATE SEGDEBUG IFACE
  AX = 4508h
  BX = ordinal (or 0000h)
  CX = segment
  DX = instance (or 0000h)
  SI = type (or 0000h)
  ES:DI -> ASCIZ module name
Notes:  this call is an alternate entry to the profiler's SEGDEBUG
    interface, but only to function 0, for notifying the profiler of
    each new segment loaded.  The SHOWHITS utility then examines the
    profiler's output files (CSIPS.DAT and SEGENTRY.DAT) in conjunction
    with symbol files to provide information in a useful form.
  this call does not have a corresponding Windows function
SeeAlso: AX=4500h
--------D-2F4601-----------------------------
INT 2F CU - MS Windows WINOLDAP - SWITCHING ???
  AX = 4601h
Return: ???
Note: the DOS 5+ kernel intercepts this function and copies the MCB
     following the caller's PSP memory block into the DOS data segment;
     in conjunction with AX=4602h, this intercept is used by DOS to
     avoid corruption of the Windows real-mode heap's end sentinel
SeeAlso: AX=1700h,AX=4602h
--------D-2F4602-----------------------------
INT 2F CU - MS Windows WINOLDAP - SWITCHING ???
  AX = 4602h
Return: ???
Note: the DOS 5+ kernel intercepts this function and copies the
    previously-saved MCB from the DOS data segment into the MCB following
    the caller's PSP memory block; in conjunction with AX=4601h, this
    intercept is used by DOS to avoid corruption of the Windows real-mode
    heap's end sentinel
SeeAlso: AX=1700h,AX=4601h
--------E-2F46-------------------------------
INT 2F U - Windows/286 DOS Extender
  AH = 46h
  AL = subfunction (03h,04h)
Return: ???
Note: these two subfunctions are called by MS Windows 3.0
--------v-2F4653CX0002-----------------------
```

```
INT 2F - F-PROT v1.x only - F-LOCK.EXE - API
  AX = 4653h ('FS')
  CX = 0002h
  BX = subfunction
       0000h  installation check
    Return: AX = FFFFh
       0001h  uninstall
    Return: AX,BX,ES destroyed
       0002h  disable (v1.08 and below only)
       0003h  enable (v1.08 and below only)
Program: F-LOCK is part of the shareware F-PROT virus/trojan protection
    package by Fridrik Skulason
SeeAlso: AX=4653h/CX=0003h,AX=CA00h,INT 21/AX=4BEEh
Index:  installation check;F-LOCK|uninstall;F-LOCK
--------v-2F4653CX0003-----------------------
INT 2F - F-PROT v1.x only - F-XCHK.EXE - API
  AX = 4653h ('FS')
  CX = 0003h
  BX = subfunction
       0000h  installation check
    Return: AX = FFFFh
       0001h  uninstall
    Return: AX,BX,ES destroyed
Program: F-XCHK is part of the shareware F-PROT virus/trojan protection
    package by Fridrik Skulason
SeeAlso: AX=4653h/CX=0002h,AX=4653h/CX=0004h,AX=CA00h
Index:  installation check;F-XCHK|uninstall;F-XCHK
--------v-2F4653CX0004-----------------------
INT 2F - F-PROT v1.x only - F-POPUP.EXE - API
  AX = 4653h ('FS')
  CX = 0004h
  BX = subfunction
       0000h  installation check
    Return: AX = FFFFh
       0001h  uninstall
    Return: AX,BX,ES destroyed
       0002h  disable (v1.08 and below only)
        display message (v1.14+)
       other registers: ???
       0003h  enable (v1.08 and below only)
        display message (v1.14+)
```

```
      other registers: ???
      Return: AX = key pressed by user
Program: F-POPUP is part of the shareware F-PROT virus/trojan protection
    package by Fridrik Skulason
SeeAlso: AX=4653h/CX=0003h,AX=4653h/CX=0005h,AX=CA00h
Index:  installation check;F-POPUP|uninstall;F-POPUP
--------v-2F4653CX0005-----------------------
INT 2F - F-PROT v1.x only - F-DLOCK.EXE - API
  AX = 4653h ('FS')
  CX = 0005h
  BX = subfunction
      0000h installation check
    Return: AX = FFFFh
      0001h uninstall
    Return: AX,BX,ES destroyed
Program: F-DLOCK is part of the shareware F-PROT virus/trojan protection
    package by Fridrik Skulason
SeeAlso: AX=4653h/CX=0004h,AX=CA00h
Index:  installation check;F-DLOCK|uninstall;F-DLOCK
----------2F4653CX0007-----------------------
INT 2F - F-PROT v2.x - VIRSTOP - ENABLE/DISABLE BOOTSECTOR READ CHECKING
  AX = 4653h ('FS')
  CX = 0007h
  BL = new state of bootsector checking (01h = disabled)
Program: VIRSTOP is the resident virus-checker from Fridrik Skulason's F-PROT
    virus/trojan protection package
----------2F4653CX0008-----------------------
INT 2F - F-PROT v2.x - ???
  AX = 4653h ('FS')
  CX = 0008h
  ???
Return: ???
Note: called by F-PROT v2.x VIRSTOP
----------2F4653CX0008-----------------------
INT 2F - F-PROT v2.x - VIRSTOP - INSTALLATION CHECK
  AX = 4653h ('FS')
  CX = 0008h
Return: AX = 5346h if installed
      BX = version???
      DS:SI -> ASCIZ name of file containing virus signatures
      DS:DI -> 80-byte buffer for ???
```

```
--------W-2F4680-----------------------------
INT 2F U - MS Windows v3.0 - INSTALLATION CHECK
  AX = 4680h
Return: AX = result
      0000h MS Windows 3.0 running in real (/R) or standard (/S) mode,
      or DOS 5 DOSSHELL active
      nonzero  no Windows, Windows prior to 3.0, or Windows3 in enhanced
      mode
Note: Windows 3.1 finally provides an installation check which works in all
    modes (see AX=160Ah)
SeeAlso: AX=1600h,AX=160Ah
----------2F47-------------------------------
INT 2F U - ???
  AH = 47h
  ???
Return: ???
Note: reportedly called by Microsoft BASIC Compiler v7.0
--------K-2F4800-----------------------------
INT 2F - DOS 5+ DOSKEY - INSTALLATION CHECK
  AX = 4800h
Return: AL = nonzero if installed
      (DOS 5.0 and 6.0 return AX=AA02h, Novell DOS 7 returns AX=EDFFh)
      ES = segment of DOSKEY resident portion
      DX = ??? (Novell DOS only; 02E6h for shipped v0.01)
Notes:  DOSKEY chains if AL is not 00h or 10h on entry
  this function is supported by Novell DOS 7 DOSKEY
SeeAlso: AX=4800h"PCED",AX=4810h
--------K-2F4800-----------------------------
INT 2F - PCED v2.1 - INSTALLATION CHECK
  AX = 4800h
Return: AX = AACDh if installed
      ES = segment of PCED kernel (PCED has multiple code segments)
Program: PCED v2.1 is a command line editor/history/macro facility by
    Cove Software.  It is the commercial version of the freeware CED.
Notes:  DOSKEY also responds to this call if installed, returning AX=AA02h.
  unlike DOSKEY, PCED does *not* chain if AL contains an
    unsupported function code.  It IRETs with all registers intact.
--------K-2F4810-----------------------------
INT 2F - DOS 5+ DOSKEY, PCED v2.1 - READ INPUT LINE FROM CONSOLE
  AX = 4810h
  DS:DX -> line buffer (see #01344 at INT 21/AH=0Ah)
```

```
Return: AX = 0000h if successful
Notes:  the first byte (length) of the buffer MUST be 80h, or MS-DOS's DOSKEY
    chains to the previous handler; PCED and Novell DOS allow sizes
    other than 80h
  if the user's input is a macro name, no text is placed in the buffer
    even though AX=0000h on return; the program must immediately issue
    this call again to retrieve the expansion of the macro.  Similarly,
    if the user enters a special parameter such as $*, this call must
    be repeated to retrieve the expansion; on the second call, DOSKEY
    overwrites the macro name on the screen with its expansion.
  unlike DOSKEY, PCED expands all macros on the first call, so it is
    not necessary to make two calls; since the buffer is not empty on
    return, DOSKEY-aware programs will not make the second call
  DOSKEY chains if AL is not 00h or 10h on entry
  this function is supported by Novell DOS 7 DOSKEY
SeeAlso: AX=4800h,INT 21/AH=0Ah
--------K-2F48C0----------------------------
INT 2F - PCED v2.1 - PCED API
  AX = 48C0h
  DX = API function code
  other registers as required by the specified function
Return: CF clear if successful
  CF set on error
      AX = PCED error code
  other registers as appropriate for API function
Program: PCED v2.1 is a command line editor/history/macro facility by
    Cove Software.  It is the commercial version of the freeware CED.
Note: the full API information is available from Cove Software
SeeAlso: AX=4800h"PCED",AX=48C1h,AX=48C2h,AX=48C3h
--------U-2F48C1BL00------------------------
INT 2F - PCED/VSTACK - INSTALLATION CHECK
  AX = 48C1h
  BL = 00h
Return: AX = 0000h if installed
      BX = VSTACK resident segment
Program: VSTACK is a resident backscroll utility included as part of the PCED
    package by Cove Software
Note: chains if BL <> 00h on entry
SeeAlso: AX=48C0h,AX=48C2h
--------U-2F48C2BL00------------------------
INT 2F - PCED/ATTRIB - INSTALLATION CHECK
```

```
  AX = 48C2h
  BL = 00h
Return: AX = 0000h if installed
      BX = ATTRIB resident segment
Program: ATTRIB is a resident file attribute changer included as part of the
    PCED package by Cove Software
Note: chains if BL <> 00h on entry
SeeAlso: AX=48C0h,AX=48C1h,AX=48C3h
--------K-2F48C3BL00-------------------------
INT 2F - PCED/KEYDEF - INSTALLATION CHECK
  AX = 48C3h
  BL = 00h
Return: AX = 0000h if installed
      BX = KEYDEF resident segment
Program: KEYDEF is a resident keyboard redefinition utility included as part
    of the PCED package by Cove Software
Note: chains if BL <> 00h on entry
SeeAlso: AX=48C0h,AX=48C2h,AX=48C4h
--------U-2F48C4BL00-------------------------
INT 2F - PCED/FLIST - INSTALLATION CHECK
  AX = 48C4h
  BL = 00h
Return: AX = 0000h if installed
      BX = FLIST resident segment
Program: FLIST is a resident filelist processor included as part of the PCED
    package by Cove Software
Note: chains if BL <> 00h on entry
SeeAlso: AX=48C0h,AX=48C3h,AX=48C5h
--------U-2F48C5BL00-------------------------
INT 2F - PCED/ASSOC - INSTALLATION CHECK
  AX = 48C5h
  BL = 00h
Return: AX = 0000h if installed
      BX = ASSOC resident segment
Program: ASSOC is a resident utility included as part of the PCED package which
    associates files with executable programs based on their extensions
Note: chains if BL > 02h on entry
SeeAlso: AX=48C0h,AX=48C4h,AX=48C5h/BL=01h,AX=48C5h/BL=02h
--------U-2F48C5BL01-------------------------
INT 2F - PCED/ASSOC - GET VERSION
  AX = 48C5h
```

```
  BL = 01h
Return: AX = 0000h if installed
      BX = binary ASSOC version (BL = major, BH = minor)
Note: chains if BL > 02h on entry
SeeAlso: AX=48C0h,AX=48C5h/BL=00h,AX=48C5h/BL=02h
--------U-2F48C5BL02------------------------
INT 2F - PCED/ASSOC - ASSOCIATION TEST
  AX = 48C5h
  BL = 02h
  DS:SI -> ASCIZ filename
Return: AX = status
      0000h if filename is unknown
      0001h if there is an association defined for the file
  BX destroyed
Program: ASSOC is a resident utility included as part of the PCED package which
    associates files with executable programs based on their extensions
Note: chains if BL > 02h on entry
SeeAlso: AX=48C0h,AX=48C5h/BL=00h,AX=48C5h/BL=01h
----------2F49------------------------------
INT 2F U - DOS 5.0+ SETUP
  AH = 49h
  AL = function
      00h update format completion gauge
    BX = percentage complete, 0000h when done, FFFFh if aborted
    Return: AX = status
          0000h continue formatting
          else  installation program wants FORMAT to abort
      10h get pointer to resident data
    AX = FFFFh if supported
        ES:BX -> internal structure
  ???
Return: ???
--------D-2F4A00CX0000----------------------
INT 2F CU - DOS 5+ - FLOPPY-DISK LOGICAL DRIVE CHANGE NOTIFICATION
  AX = 4A00h
  CX = 0000h
  DH = new drive number
  DL = current drive number
Return: CX = FFFFh to skip "Insert diskette for drive X:" message
Note: called by MS-DOS 5.0+ IO.SYS just before displaying the message
    "Insert diskette for drive X:" on single-floppy systems
```

```
--------D-2F4A01-----------------------------
INT 2F - DOS 5+ - QUERY FREE HMA SPACE
  AX = 4A01h
Return: BX = number of bytes available in HMA (0000h if DOS not using HMA)
  ES:DI -> start of available HMA area (FFFFh:FFFFh if not using HMA)
Notes:  called by Windows 3.1 DOSX.EXE
  supported by Novell DOS 7
SeeAlso: AX=4310h,AX=4A02h
--------D-2F4A02-----------------------------
INT 2F - DOS 5+ - ALLOCATE HMA SPACE
  AX = 4A02h
  BX = number of bytes
Return: ES:DI -> start of allocated HMA block or FFFFh:FFFFh
  BX = number of bytes actually allocated (rounded up to next paragraph
       for DOS 5.0 and 6.0)
Notes:  this call is not valid unless DOS is loaded in the HMA (DOS=HIGH)
  called by Windows 3.1 DOSX.EXE
  supported by Novell DOS 7
SeeAlso: AX=4A01h,AX=4A03h
--------D-2F4A03-----------------------------
INT 2F U - Windows95 - DOS KERNEL - (DE)ALLOCATE HMA MEMORY BLOCK
  AX = 4A03h
  CX = segment of block's owner???
  DL = subfunction
      00h allocate block
    BX = number of bytes
    Return: DI=FFFFh if unable to allocate
      ES:DI -> allocated block
      01h resize block
    ES:DI -> previously-allocated block
    BX = new size in bytes (must be less than original size???)
    Return: DI=FFFFh if unable to allocate
      ES:DI -> reallocated block
    Note: the contents of the original block are NOT copied
      02h free block???
    ES:DI -> block to be freed
Note: in MS-DOS 7.x, function 4A02h is implemented by calling this function
    with DL=00h
SeeAlso: AX=4A02h
--------D-2F4A04-----------------------------
INT 2F U - Windows95 - DOS KERNEL - GET START OF HMA MEMORY CHAIN
```

```
  AX = 4A04h
Return: AX = 0000h if function supported
      ES:DI -> first HMA memory control block (see #02800)

Format of Windows95 HMA memory control block:
Offset  Size  Description (Table 02800)
 00h  2 BYTEs signature "MS" (4Dh 53h)
 02h  WORD  segment of owner (or segment at which to address block???)
    0000h = free
    0001h = DOS???
    FF33h = IO.SYS
    FFFFh = MSDOS.SYS
 04h  WORD  size of memory block (not including this header)
 06h  WORD  offset of next memory block in segment FFFFh, or 0000h if last
 08h  8 BYTEs unused (explicitly set to 0 for MS-DOS 7.10)
--------T-2F4A05-----------------------------
INT 2F U - DOS 5+ DOSSHELL - TASK SWITCHING API???
  AX = 4A05h
  SI = function
      0000h reset???
      0001h ???
    ES:BP -> 80-byte buffer containing ???
      0002h ???
      0003h ???
      0004h ???
    BL = ???
      0005h ???
      0006h get ???
    Return: ES:SI -> ???
      0007h get ???
    Return: AX = ???
      0008h get ???
    Return: DX:AX -> ??? (internal control data of some kind)
      0009h get ???
    Return: ES:SI -> ??? (apparently identical to function 0006h)
      000Ah ???
    BL = length of buffer
    ES:BP -> buffer containing ???
      000Bh get ???
    Return: AX = ???
      000Ch ???
```

```
    BL = ???
    Return: if BL nonzero on entry
          DX:AX -> ???
      if BL = 00h on entry
          ES:SI -> ???
Notes:  DOSSHELL chains to the previous handler if SI is not one of the values
    listed above
  the DOSSWAP.EXE module calls functions 03h,04h,05h,07h,08h,09h,0Ch
  the Windows 3.1 DSWAP.EXE and WSWAP.EXE task switchers use these calls
SeeAlso: AX=4B01h
--------D-2F4A06-----------------------------
INT 2F CU - DOS 5+ - DOS SUPERVISOR "REBOOT PANEL" - ADJUST MEMORY SIZE
  AX = 4A06h
  DX = segment following last byte of conventional memory
Return: DX = segment following last byte of memory available for use by DOS
Desc: used to override the default memory size when booting diskless
    workstations
Notes:  called by MS-DOS 5+ IO.SYS and DR DOS 6.0+ IBMBIO.OCM startup code if
    the signature "RPL" is present three bytes beyond the INT 2F handler;
    this call overrides the value returned by INT 12
  hooked by RPL code at the top of memory to protect itself from being
    overwritten; DOS builds a memory block with owner = 0008h and name
    "RPL" which must be freed by the RPL code when it is done.
    Under DR DOS, it is sufficient to set the owner field of the MCB to
    0000h.
  In addition to the test for "RPL", DR PalmDOS (since 1992/08/25),
    DR DOS 6.0 "Business update March 1993", DR DOS "Panther" and
    "StarTrek", and Novell DOS 7+ also check for a "RPLOADER" signature.
    If this 2nd signature is found, IBMBIO.COM will store the INT 2Fh
    vector for later use after the BIOS init, when at several points it
    directly calls the RPLOADER via an emulated INT 2Fh with
    AX=12FFh/BX=5/CX=0/DX=1 and a phase code 1, 2 or 3 on the stack.
    This permits the RPLOADER to keep track of the initialization
    process and clean or fix up anything it wishes.  The "phase 1"
    broadcast is issued after the BIOS init code and data have been
    relocated (e.g. into the HMA), "phase 2" gets issued immediately
    before the CONFIG.SYS processing begins and the DOS code and data
    are relocated, and the closing "phase 3" happens to permit any
    final tidy ups before the memory manager gets acknowledgement of
    completion.
SeeAlso: INT 12"BIOS",INT 18"BOOT HOOK",AX=4A07h,INT 2F/AX=12FFh/BX=0005h
```

```
--------N-2F4A07-----------------------------
INT 2F U - RESERVED FOR PROTMAN SUPPORT
  AX = 4A07h
  ???
Return: ???
SeeAlso: AX=4A06h,INT 18"BOOT HOOK"
--------c-2F4A10BX0000----------------------
INT 2F - SMARTDRV v4.00+ - INSTALLATION CHECK AND HIT RATIOS
  AX = 4A10h
  BX = 0000h
  CX = EBABh (v4.1+; see Note), and CX <> 0EDCh
Return: AX = BABEh if installed
      DX:BX = cache hits
      DI:SI = cache misses
      CX = number of dirty cache elements
      BP = version in BCD (4.10 = 0410h)
    4.0  = 0400h (Windows 3.1)
    4.01 = 0401h (MS-DOS 6.0, 1992/07/30)
    4.10 = 0410h (1992/11/11)
    4.20 = 0420h
    5.0  = 0500h (MS-DOS 6.2)
Notes:  most of the SMARTDRV API, including this call, is supported by
    PC-Cache v8.0 and recent versions of the Norton Caches
  the internal name for SMARTDRV is "BAMBI", hence the magic "BABE".
  if DBLSPACE.BIN is installed but SMARTDRV has not yet been installed,
    then calls of this function with CX<>EBABh on entry cause
    DBLSPACE.BIN to display the error message
    "Cannot run SMARTDrive 4.0 with DoubleSpace" and abort the caller
    with INT 21/AX=4C00h
  Since the Novell DOS 7 - DR-DOS 7.03 NWCACHE supports a similar
    install check at this function, but the returned registers
    contain different data, the caller must take the returned AX value
    into account to ensure proper interpretation of the returned info.
    The caller should ensure CX <> 0EDCh to avoid any interaction
    with the NWCACHE and COMMAND.COM special case of CX=0EDCh.
    Although this is an install check, NWCACHE always flushes delayed
    writes to disk, when calling this function with CX<>0EDCh or while
    /FLUSH:ON is active, but this is nothing to rely upon. The reason
    for this could be to get more accurate statistics, while SMARTDRV
    just takes a on-the-fly snapshot of the statistic vars.
  The private NWCACHE/NLCACHE install check INT2Fh/D8C0h does not flush
```

```
    to disk, and therefore can be used by callers that must ensure
    the buffers aren't flushed on this function (e.g. alternative
    command shells may need this to make their "flush before prompt"
    feature configurable).
  The NWCACHE 0EDCh special case will most probably vanish with
    future issues of NWCACHE (2.0+).
  SMARTDRV v3.x had a completely different API using IOCTL calls, which
    was also supported by the Norton Caches
BUG:  Although DS and ES registers are listed as unmodified, some
    releases of SMARTDRV seem to trash the DS register, at least the
    MS-DOS 6.2+ KEYB takes care to preserve the DS register when calling
    this function.
SeeAlso: AX=4A10h/BX=0001h,AX=4A10h/BX=0004h,AX=4A10h/BX=0005h
SeeAlso: AX=4A10h/BX=0007h,AX=4A10h/BX=1234h,AX=4A11h/BX=0000h
SeeAlso: INT 21/AX=4402h"SMARTDRV",INT 21/AX=4403h"SMARTDRV"
--------c-2F4A10BX0000-----------------------
INT 2F U - Novell NWCACHE - INSTALLATION CHECK AND STATISTICS
  AX = 4A10h
  BX = 0000h
  CX = magic value
       0EDCh flush delayed writes
    ('EDC' = Novell European Development Center)
       other (use EBABh to avoid SMARTDRV/DBLSPACE interaction)
Return: (also refer to notes below)
  AX = 6756h ('gV') if installed
       CF clear
       BX = cache memory type
    01h conventional memory
    02h extended memory
    03h EMS
       DI:SI = cache misses
       CX:DX = segment:offset address of data area containing statistics
       (see #04095)
       BP = version in BCD
    0100h = 1.00
    0101h = 1.01 & 1.02 (!)
Notes:  SMARTDRV 4.0+ uses a similar installation check on this function but
    returns different data.  To ensure proper interpretation of
    the returned values, the caller must check the returned magic
    value in AX. Since SMARTDRV may also alter DI:SI (and possibly
    DS), the caller should take care of this, too.
```

```
  The Novell DOS / DR-DOS 7.x COMMAND.COM invokes this function with
    CX=0EDCh, which causes NWCACHE to flush any delayed writes before
    returning the normal register values if it is configured to flush
    writes before returning to the DOS prompt (/FLUSH:ON); however, if
    /FLUSH:OFF, this function becomes a NOP with CX=0EDCh.
  Note that for this function the 0EDCh special case may vanish with
    future issues of NWCACHE (2.0+), and the function would then always
    do the normal install check, and not flush the cache.
SeeAlso: AX=4A10h/BX=0001h"NWCACHE"

Format of NWCACHE statistics:
Offset  Size  Description (Table 04095)
 00h  DWORD number of read requests
 04h  DWORD number of reads performed
 08h  DWORD number of write requests
 0Ch  DWORD number of writes performed
 10h  WORD  number of disk errors
 12h  WORD  number of cache memory errors
--------c-2F4A10BX0001-----------------------
INT 2F - SMARTDRV v4.00+ - FLUSH BUFFERS (COMMIT CACHE)
  AX = 4A10h
  BX = 0001h
Note: this function is also supported by PC-Cache v8.0.
SeeAlso: AX=4A10h/BX=0000h,AX=4A10h/BX=0002h,INT 21/AH=0Dh
--------c-2F4A10BX0001-----------------------
INT 2F U - Novell NWCACHE - ???
  AX = 4A10h
  BX = 0001h
  CX = 0EDCh ('EDC' = Novell European Development Center)
Return: ???
SeeAlso: AX=4A10h/BX=0000h"NWCACHE"
--------c-2F4A10BX0002-----------------------
INT 2F - SMARTDRV v4.00+ - RESET CACHE
  AX = 4A10h
  BX = 0002h
Note: this function is also supported by PC-Cache v8.0.
SeeAlso: AX=4A10h/BX=0000h,AX=4A10h/BX=0001h
--------c-2F4A10BX0003-----------------------
INT 2F - SMARTDRV v4.00+ - STATUS
  AX = 4A10h
  BX = 0003h
```

```
  BP = drive number (0=A, 1=B, etc.)
    FFFFh = all drives (NWCACHE only)
  DL = subfunction
       00h only get information
       01h turn on read cache
       02h turn off read cache
       03h turn on write cache
       04h turn off write cache
       ---NWCACHE---
       05h reduce cache size
    CX = number of KB to reduce (can't shrink below min size)
       06h increase cache size
    CX = number of KB to increase (can't grow above max size)
Return: AX = BABEh if OK
  DL = status (see #02801)
  DL = FFh if drive does not exist
Notes:  If the read cache is off, reads will not be cached, but writes will
    continue to be cached if the write-cache is enabled.
  this function is also supported by PC-Cache v8.0.
  SMARTDRV flushes the cache if sub-functions 02h or 04h have been
    called to disable read or write caching.
  On calling this function, NWCACHE flushes any delayed writes,
    but you should not rely on this, since this behavior may vanish
    in future versions.
BUG:  The MS-DOS 6.20+ SCANDISK disables a SMARTDRV 4+ compatible cache
    for the drive it is about to process.  However, at least when the
    cache loaded is NWCACHE, the cache will still remain disabled for
    this drive after SCANDISK has finished.  The reason for this behavior
    is not yet known. The workaround is to re-enable with the
    NWCACHE d:+ syntax afterwards, where d: is the corresponding
    drive letter.
SeeAlso: AX=4A10h/BX=0000h

Bitfields for SMARTDRV status:
Bit(s)  Description (Table 02801)
 7  not cached (no read-caching)
 6  write-through (not write-cached)
 0-5  real drive number (0=A, 1=B...)
Notes:  the "real" drive number is always set to 00h for RAM disks and to 3Fh
    for CD-ROMs; unfortunately, the latter means that an uncached CD-ROM
    has status FFh, just as a nonexistent drive does
```

```
  also supported by Novell DOS 7 - Caldera/Lineo DR-DOS 7.03 NWCACHE
--------c-2F4A10BX0004-----------------------
INT 2F - SMARTDRV v4.00+ - GET CACHE SIZE
  AX = 4A10h
  BX = 0004h
Return: AX = size in elements of full-sized cache
  BX = current size in elements
  CX = size of one element in bytes
  DX = number of elements under Windows
Note: this function is also supported by PC-Cache v8.0.
SeeAlso: AX=4A10h/BX=0000h,AX=4A10h/BX=0003h,AX=4A10h/BX=0005h
--------c-2F4A10BX0005-----------------------
INT 2F - SMARTDRV v4.00+ - GET DOUBLE-BUFFER STATUS
  AX = 4A10h
  BX = 0005h
  BP = drive number (0=A, 1=B...)
Return: AX = BABEh if double-buffered
      ES:DI -> 16-byte array of status bytes for fixed disks (see #02802)
SeeAlso: AX=4A10h/BX=0000h,AX=4A10h/BX=0003h,AX=4A10h/BX=0006h

(Table 02802)
Values for SMARTDRV status byte:
 00h  state unknown
 FFh  drive double-buffered
 else not double-buffered
--------c-2F4A10BX0006-----------------------
INT 2F CU - SMARTDRV v4.00+ - CHECK IF DRIVE CACHEABLE
  AX = 4A10h
  BX = 0006h
  CL = drive number (01h = A:)
Return: AX = 0006h if drive should not be cached by SMARTDRV
Note: called by SMARTDRV at startup to determine whether it should cache
    a particular drive
SeeAlso: AX=4A10h/BX=0000h
--------c-2F4A10BX0007-----------------------
INT 2F - SMARTDRV v4.00+ - GET DEVICE DRIVER FOR DRIVE
  AX = 4A10h
  BX = 0007h
  BP = drive number (00h=A:)
Return: DL = unit number within device driver
  ES:DI -> device driver header for drive (see #01646)
```

```
Note: this function is also supported by PC-Cache v8.0.
  this call is reported to always return the driver header of the
    standard block driver (A:-C:+) for SmartDrive v5.00 from MS-DOS 6.2
SeeAlso: AX=4A10h/BX=0000h,AX=4A11h/BX=0003h,AX=4A11h/BX=0004h
--------c-2F4A10BX0008-----------------------
INT 2F - SMARTDRV v4.20+ - GET/SET FLUSH BEFORE PROMPT, CD-ROM SUPPORT
  AX = 4A10h
  BX = 0008h
  DL = subfunction
      00h set
    DH = new states
        bit 0: flush before prompt
        bits 1-7 reserved (0)
      01h get
    Return: DH = status flags
          bit 0: (v4.2+) flush before prompt
          bit 1: (v5.0+) CD-ROM caching support installed
Note: v4.2 was an interim release to fix problems in the SMARTDRV included
    with MS-DOS 6.00; v5.00 is included with MS-DOS 6.2
--------c-2F4A10BX000A-----------------------
INT 2F - SMARTDRV v4.00+ - GET ELEMENT STATUS TABLE
  AX = 4A10h
  BX = 000Ah
Return: ES:BX -> information pointer table (see #02803)
Note: this function is also supported by PC-Cache v8.0.
SeeAlso: AX=4A10h/BX=0000h

Format of SMARTDRV information pointer table:
Offset  Size  Description (Table 02803)
 00h  WORD  offset of ??? byte/word array (byte if elements < 2000h bytes)
 02h  WORD  offset of dirty flag byte/word array (byte if elts < 2000h)
    each byte/word is a bit string of the dirty sectors in element
 04h  WORD  offset of word array containing low halves of unique
      identifiers for the corresponding element's contents
 06h  WORD  offset of word array containing high halves of unique
      identifiers for the corresponding element's contents
 08h  WORD  offset of WORD containing current number of elements in cache
--------c-2F4A10BX1234-----------------------
INT 2F - SMARTDRV v4.00+ - SIGNAL SERIOUS ERROR
  AX = 4A10h
  BX = 1234h
```

```
Desc: this function pops up a message box saying that a serious error
    occurred and to hit R to retry, then waits for the keypress
Note: this function is also supported by PC-Cache v8.0.
SeeAlso: AX=4A10h/BX=0000h
--------k-2F4A11BX0000-----------------------
INT 2F - DBLSPACE.BIN - "GetVersion" - INSTALLATION CHECK
  AX = 4A11h
  BX = 0000h
Return: AX = 0000h (successful)
  BX = 444Dh ("DM")
  CL = first drive letter used by DBLSPACE (41h ['A'] = A:)
  CH = number of drive letters used by DBLSPACE
  DX = internal DBLSPACE.BIN version number (bits 14-0)
    bit 15 set if DBLSPACE.BIN has not yet been relocated to final
      position in memory (i.e. DBLSPACE.SYS /MOVE)
Program: DBLSPACE.BIN is the resident driver for DoubleSpace, the
    disk-compression software bundled with MS-DOS 6.0 and 6.20
Notes:  this function is also supported by the version of SuperStor bundled
    with IBM DOS 6.x and "preloading" versions of Stacker
  the complete DBLSPACE.BIN API is still supported by MS-DOS 6.22's
    DRVSPACE.BIN
SeeAlso: AX=4A11h/BX=0001h,AX=4A11h/BX=0002h,AX=4A11h/BX=0003h
SeeAlso: AX=4A11h/BX=0005h,AX=4A11h/BX=0007h,AX=4A11h/BX=FFFFh
SeeAlso: INT 21/AX=4404h"DBLSPACE"
--------k-2F4A11BX0001-----------------------
INT 2F - DBLSPACE.BIN - "GetDriveMapping" - GET DRIVE MAPPING
  AX = 4A11h
  BX = 0001h
  DL = drive number (0=A:)
Return: AX = status (see also #02804)
      0000h successful
    if DL was compressed drive,
        BL = host drive (bit 7 set if drive is compressed)
    else if DL was host drive,
        BL = compressed drive
    else
        BL = specified drive (if available for DoubleSpace)
    BH = DoubleSpace sequence number
      other error code (0101h) (see #02804)
    apparently never returned for the MS-DOS 6.2 DoubleSpace
Note: the compressed volume file for the specified compressed drive is
```

```
    host:\DBLSPACE.sequence
SeeAlso: AX=4A11h/BX=0000h

(Table 02804)
Values for DBLSPACE function status:
 0000h  successful
 0100h  bad function
 0101h  invalid drive
 0102h  not a compressed drive
 0103h  drive already swapped
 0104h  drive not swapped
SeeAlso: #02806
--------k-2F4A11BX0002-----------------------
INT 2F - DBLSPACE.BIN - "Swap Drive" - SWAP DRIVE LETTERS OF CVF AND HOST DRIVE
  AX = 4A11h
  BX = 0002h
  DL = drive number (0=A:) of compressed drive to swap with its host
Return: AX = status (0000h,0101h,0102h,0103h) (see #02804)
Note: this function is intended for use by DBLSPACE.EXE only
SeeAlso: AX=4A11h/BX=0000h
--------k-2F4A11BX0003-----------------------
INT 2F - DBLSPACE.BIN - "DSGetEntryPoints" - GET DEVICE DRIVER ENTRY POINTS
  AX = 4A11h
  BX = 0003h
  CL = drive number (0=A:) of compressed drive
Return: CL = FFh on error (not compressed drive)
  CL <> FFh driver unit number of host drive
      ES:SI -> device driver's strategy routine
      ES:DI -> device driver's interrupt routine
  BX destroyed
Note: in conjunction with subfunction 0004h, this call allows disk caches
    like SMARTDRV to apply a device driver wrapper to DoubleSpaced
    drives just like SMARTDRV applies to regular block devices
SeeAlso: AX=4A10h/BX=0007h,AX=4A11h/BX=0000h,AX=4A11h/BX=0004h
--------k-2F4A11BX0004-----------------------
INT 2F - DBLSPACE.BIN - "DSSetEntryPoints" - SET DEVICE DRIVER ENTRY POINTS
  AX = 4A11h
  BX = 0004h
  CL = drive number (0=A:) of compressed drive
  DL = unit number for new driver entry points
  DH = 00h
```

```
  ES:SI -> device driver strategy routine to call for drive
  ES:DI -> device driver interrupt routine to call for drive
Return: CL = FFh on error (not a compressed drive)
  BX destroyed
Program: DBLSPACE.BIN is the resident driver for DoubleSpace, the
    disk-compression software bundled with MS-DOS 6.0
Note: in conjunction with subfunction 0003h, this call allows disk caches
    like SMARTDRV to apply a device driver wrapper to DoubleSpaced
    drives just like SMARTDRV applies to regular block devices
SeeAlso: AX=4A10h/BX=0007h,AX=4A11h/BX=0000h,AX=4A11h/BX=0003h
--------k-2F4A11BX0005-----------------------
INT 2F - DBLSPACE.BIN - "ActivateDrive" - MOUNT COMPRESSED DRIVE
  AX = 4A11h
  BX = 0005h
  DL = drive number (0=A:) to assign to new drive
  ES:SI -> activation record (see #02805)
Return: status returned in activation record (see #02806)
SeeAlso: AX=4A11h/BX=0000h,AX=4A11h/BX=0006h

Format of DBLSPACE activation record:
Offset  Size  Description (Table 02805)
 00h  2 BYTEs signature "MD" (4Dh 44h)
 02h  BYTE  4Dh ('M') mount command
 03h  BYTE  error code (set to FFh before calling) (see #02806)
 04h  BYTE  host drive number (0=A:)
 05h  ??? DISK_UNIT structure (not documented)

(Table 02806)
Values for DBLSPACE Mount error code:
 00h  successful
 01h  drive letter not available for DoubleSpace
 02h  drive letter already in use
 03h  no more disk units (increase MaxRemovableDrives in .INI)
 09h  CVF too fragmented
SeeAlso: #02805,#02804
--------k-2F4A11BX0006-----------------------
INT 2F - DBLSPACE.BIN - "DeactivateDrive" - UNMOUNT COMPRESSED DRIVE
  AX = 4A11h
  BX = 0006h
  DL = drive number (0=A:) to unmount
Return: AX = status (0000h,0102h) (see #02804)
```

```
SeeAlso: AX=4A11h/BX=0000h,AX=4A11h/BX=0005h
--------k-2F4A11BX0007-----------------------
INT 2F - DBLSPACE.BIN - "GetDriveSpace" - GET SPACE AVAIL ON COMPRESSED DRIVE
  AX = 4A11h
  BX = 0007h
  DL = compressed drive number (0=A:)
Return: AX = status (0000h,0102h) (see also #02804)
      0000h successful
    DS:SI -> free space record (see #02807)
Program: DBLSPACE.BIN is the resident driver for DoubleSpace, the
    disk-compression software bundled with MS-DOS 6.0
SeeAlso: AX=4A11h/BX=0000h,AX=4A11h/BX=0008h

Format of DBLSPACE free space record:
Offset  Size  Description (Table 02807)
 00h  DWORD total number of sectors in drive's sector heap
 04h  DWORD number of free sectors in drive's sector heap
--------k-2F4A11BX0008-----------------------
INT 2F - DBLSPACE.BIN - "GetFileFragmentSpace" - GET SIZE OF FRAGMENT HEAP
  AX = 4A11h
  BX = 0008h
  DL = compressed drive number (0=A:)
Return: AX = status (0000h,0102h) (see also #02804)
      0000h successful
    BX = maximum entries in File Fragment heap
    CX = available entries in File Fragment heap
SeeAlso: AX=4A11h/BX=0000h,AX=4A11h/BX=0007h,AX=4A11h/BX=0009h
--------k-2F4A11BX0009-----------------------
INT 2F - DBLSPACE.BIN - "GetExtraInfo" - DETERMINE NUMBER OF DISK_UNIT STRUCTS
  AX = 4A11h
  BX = 0009h
  DL = compressed drive number (0=A:)
Return: AX = status (see also #02804)
      0000h successful
    CL = number of DISK_UNIT structures allocated
        (see AX=4A11h/BX=0005h)
    CH = DoubleGuard enabled-checks bitflags in bits 6-0 (DOS 6.2)
Note: the DoubleGuard checks are enabled or disabled as a block by the
    DoubleGuard= line in DBLSPACE.INI; they may be individually set with
    the CheckSum= line.
SeeAlso: AX=4A11h/BX=0000h,AX=4A11h/BX=0008h
```

```
--------k-2F4A11BX000A-----------------------
INT 2F - DBLSPACE.BIN v6.2 - SET AUTOMOUNT DRIVES
  AX = 4A11h
  BX = 000Ah
  CX:DX = bitmask of drives on which to enable AutoMount
    (DX bit 0 = A:, CX bit 0 = P:, etc.)
Return: AX = 0000h if supported
      CX:DX = old mask or 0000h:0000h on error
  BX destroyed
SeeAlso: AX=4A11h/BX=000Bh
--------k-2F4A11BX000B-----------------------
INT 2F - DBLSPACE.BIN v6.2 - GET AUTOMOUNT DRIVES
  AX = 4A11h
  BX = 000Bh
Return: AX = 0000h if supported
      CX:DX = mask of drives with AutoMount enabled
    0000h:0000h on error
  BX destroyed
SeeAlso: AX=4A11h/BX=000Ah
--------k-2F4A11BXFFFE-----------------------
INT 2F U - DBLSPACE.BIN - RELOCATE
  AX = 4A11h
  BX = FFFEh
  ES = segment to which to relocate DBLSPACE.BIN
Return: ???
Notes:  called by DBLSPACE.SYS to relocate DBLSPACE.BIN to its final position
    in memory
  this function also unhooks and discards the code providing this
    function and AX=4A11h/BX=FFFFh
  this function is also supported by the version of SuperStor bundled
    with IBM DOS 6.x and "preloading" versions of Stacker
SeeAlso: AX=4A11h/BX=FFFFh
--------k-2F4A11BXFFFF-----------------------
INT 2F U - DBLSPACE.BIN - GET RELOCATION SIZE
  AX = 4A11h
  BX = FFFFh
Return: AX = number of paragraphs needed by DBLSPACE.BIN
Note: used by DBLSPACE.SYS to relocate the DBLSPACE driver to its final
    position in memory
  this function is also supported by the version of SuperStor bundled
    with IBM DOS 6.x and "preloading" versions of Stacker
```

```
SeeAlso: AX=4A11h/BX=0000h,AX=4A11h/BX=FFFEh
--------k-2F4A12CX4D52-----------------------
INT 2F - Microsoft Realtime Compression Interface (MRCI) - RAM-BASED SERVER
  AX = 4A12h
  CX = 4D52h ("MR")
  DX = 4349h ("CI")
Return: CX = 4943h ("IC") if installed
  DX = 524Dh ("RM") if installed
       ES:DI -> MRCINFO structure (see #00725 at INT 1A/AX=B001h)
Note: this call is functionally identical to INT 1A/AX=B001h, but should be
    called first, as the latter call is used for the first, ROM-based
    MRCI server, while this call is used for RAM-based servers which
    may be partially or entirely replacing a prior server
SeeAlso: AX=4A12h/CX=5354h,INT 1A/AX=B001h
--------k-2F4A12CX5354-----------------------
INT 2F - Stacker 4 LZS Compression Interface (LZSAPI)
  AX = 4A12h
  CX = 5354h ("ST")
  DX = 4143h ("AC")
Return: AX = 4F4Bh ("OK") if installed
  CX = 7374h ("st") if installed
  DX = 6163h ("ac") if installed
       ES:DI -> entry point in LZSAPI server (usually, driver
           STACKER.COM) containing far address of an actual
           LZSINFO structure (see #02808)
SeeAlso: AX=4A12h/CX=4D52h,INT 25/AX=CDCDh

Format of LZSINFO structure:
Offset  Size  Description (Table 02808)
 00h  6 BYTEs signature "LZSAPI"
 06h  WORD  server version (200 (0C8h) for Stacker 4 and Stacker Anywhere)
 08h  4 BYTEs vendor signature
    "STAC" = Stac Electronics, Inc.
 0Ch  6 BYTEs ???
 12h  WORD  bit flags: server status/capabilities (see #02809)
 14h  DWORD -> Stacker 3.X-compatible compression procedure
 18h  DWORD -> Stacker 3.X-compatible decompression procedure
 1Ch  4 BYTEs reserved (always set to 0)
 20h  DWORD -> incremental Stacker 3.x-compatible compression procedure
      (see #02810)
 24h  4 BYTEs ???
```

```
 28h  DWORD -> incremental Stacker 3.x-compatible decompression procedure
      (see #02811)
 2Ch  4 BYTES ???
 30h  DWORD -> new (Stacker 4) compression procedure
 34h  4 BYTEs ???
 38h  DWORD -> new (Stacker 4) decompression procedure
Notes:  Stacker 4 usally keeps two different data-compression algorithms in
    memory (preferably in XMA):
      1) new (Stacker 4) data compression algorithm (4K LZ77 with hashing
      and static Huffman encoding), and
      2) old (Stacker 3.x-compatible) one (2K LZ77 with hashing), which
      is used to work with STACVOL files created under older
      versions of Stacker.

Bitfields for LZSAPI capabilities:
Bit(s)  Description (Table 02809)
 0  ???
 1  busy/error flag
 2-10 ???
 11 maximum compressor/decompressor presented

(Table 02810)
Call Stacker 3.x-style non-incremental functions with:
  STACK:  DWORD return address for compress/decompress procedure
    WORD  compression algorithm parameters (see #02812)
    WORD  size of destination buffer (in bytes)
    DWORD address of destination buffer
    WORD  size of source buffer (in bytes)
    DWORD address of source buffer
Return: (compression/decompression procedure)
  AX = size of resulting data in destination buffer
       0000h if error (either destination buffer too small or error in
       compressed data)

(Table 02811)
Call Stacker 4-style incremental procedures with:
  STACK:  DWORD return address for compr./decompr. procedure
    DWORD address of LZSSWAP structure (see #02813)
      if 0000:0000, procedure uses non-incremental technique
    WORD  compression algorithm parameters (see #02812)
    WORD  size of destination buffer (in bytes)
```

```
    DWORD address of destination buffer
    WORD  size of source buffer (in bytes)
    DWORD address of source buffer

(Table 02812)
Values for Compression algorithm parameters:
Value Compression level (/P=xx parameter)
 07F9h    1
 0621h    2
 0625h    3
 0665h    4
 0669h    5
 06E9h    6
 06EDh    7
 07D1h    8
 07D9h    9

Format of LZSSWAP structure:
Offset  Size  Description (Table 02813)
 00h  2 BYTEs signature "CS"
 02h  6 BYTEs reserved
 08h  DWORD address of destination buffer swapping procedure
 0Ch  DWORD address of stack swapping procedure
--------k-2F4A13-----------------------------
INT 2F U - DBLSPACE.BIN - DBLSPACE/MRCI STEALTH PACKET API
  AX = 4A13h
Return: AX = 134Ah if supported
      ES:BX -> entry point record (see #02814)
SeeAlso: AX=4A11h/BX=0000h,AX=4A12h/CX=4D52h"MRCI"

Format of DBLSPACE entry point record:
Offset  Size  Description (Table 02814)
 00h  DWORD pointer to FAR function for ???
 04h  5 BYTEs FAR JUMP instruction to ???
----------2F4A15BX0000----------------------
INT 2F - MS EMM386.EXE v4.46+ - INSTALL I/O VIRTUALIZATION HANDLER
  AX = 4A15h
  BX = 0000h (function number)
  DX = starting I/O address
  EDX high word = ending I/O address
  CX = number of ports to trap
```

```
  DS:SI -> I/O dispatch table (see #02815)
  DI = size of client's code and data (size of DS segment which must be
       made available to I/O dispatch function in protected mode)
Return: CF clear if successful
  CF set on error
Notes:  this interface is only available in virtual-86 mode; the I/O handlers
    will be called in protected mode
  only ports 0100h-FFFFh may be trapped; EMM386 reserved ports 0000h-
    00FFh

Format of EMM386 I/O dispatch table [array]:
Offset  Size  Description (Table 02815)
 00h  WORD  I/O port number
 02h  WORD  offset of I/O handler for port (see #02816)

(Table 02816)
Values EMM386 I/O dispatch function is called with:
  CX = Ring0 code selector for I/O handler's segment
  DS = Ring0 data selector for I/O handler's segment (alias of CS)
  EDX = faulting I/O address
  ECX = direction (00000008h for byte output, 00000000h for byte input)
    (reportedly 00h for byte/word input, 04h for byte/word output
      under DOS 6.22 EMM386)
  EAX = data in/out
Return: (via FAR RET)
  CF clear if I/O access successfully virtualized
  CF set if access not virtualized (default handler will be called to
       perform the I/O)
BUG:  32-bit I/O on trapped ports hangs the DOS 6.22 EMM386
SeeAlso: #02815
--------D-2F4A16-----------------------------
INT 2F U - Windows95 - OPEN BOOT LOG
  AX = 4A16h
Return: AX = status
      0000h successful
      FFFFh boot log file already open
      else  DOS error code
  BX destroyed
SeeAlso: AX=4A17h,AX=4A18h
--------D-2F4A17-----------------------------
INT 2F U - Windows95 - WRITE TO BOOT LOG
```

```
  AX = 4A17h
  CX = number of bytes to write
  DS:DX -> message to write (must include CR-LF if it is desired)
Return: AX = status
      0000h successful
      FFFFh boot log file not open
      else  DOS error code
Note: calls the code for INT 2F/AX=4A21h after writing to the file
SeeAlso: AX=4A17h,AX=4A18h,AX=4A21h
--------D-2F4A18-----------------------------
INT 2F U - Windows95 - CLOSE BOOT LOG
  AX = 4A18h
Return: AX = status
      0000h successful
      FFFFh boot log file not open
      else  DOS error code from closing file
  BX destroyed
SeeAlso: AX=4A16h,AX=4A17h
--------D-2F4A21-----------------------------
INT 2F U - Windows95 - ???
  AX = 4A21h
Return: AX destroyed
Note: calls INT 21/AX=4404h"IOCTL" with a five-byte buffer containing "MDF??"
SeeAlso: AX=4A17h
--------D-2F4A31-----------------------------
INT 2F U - Windows95 - ???
  AX = 4A31h
  CL = new value for ???
  DS:SI -> BYTE to be set to CL
Return: nothing
--------D-2F4A32-----------------------------
INT 2F U - Windows95 - PATCH ???
  AX = 4A32h
  BL = subfunction
      00h get ???
    Return: AX = flag: subfunction 04h has been used (0000h/FFFFh)
      DX = ??? (0000h/?)
      01h patch ??? in IO.SYS (segment 0070h)
      02h unpatch ??? in IO.SYS
      03h ???
      04h set ???, then do subfunction 01h
```

```
      05h unset ???, then do subfunction 02h
      else
    Return: nothing
--------D-2F4A33-----------------------------
INT 2F - Windows95 - CHECK MS-DOS VERSION 7
  AX = 4A33h
Return: AX = 0000h for MS-DOS 7.00+
      (officially) BX,DX,SI,DS may be destroyed
      (undoc) DS:DX -> ASCIZ primary shell executable name
      (undoc) DS:SI -> CONFIG.SYS SHELL= command line (counted string)
      (undoc) BH = ??? (0000h)
      (undoc) BL = ??? (0000h)
  AX nonzero (usually 4A33h) if MS-DOS 6- or other DOS
SeeAlso: AX=1611h,INT 21/AH=30h
--------N-2F4B-------------------------------
INT 2F - LAN Manager 2.0 DOS Enh NETWKSTA.EXE - NETWORK WORKSTATION REDIRECTOR
  AH = 4Bh
  ???
Return: ???
Note: LAN Manager enhanced mode adds features beyond the standard redirector
    file/printer services
SeeAlso: AX=118Ah,AX=4100h,AH=42h
--------T-2F4B01-----------------------------
INT 2F C - DOS 5+ TASK SWITCHER - BUILD CALLOUT CHAIN
  AX = 4B01h
  CX:DX -> task switcher entry point (see #02819)
  ES:BX = 0000h:0000h
Return: ES:BX -> callback info structure (see #02817) or 0000h:0000h
Notes:  called by the task switcher
  this function is hooked by clients which require notification of task
    switcher activities; the call must first be passed on to the prior
    handler with registers unchanged using a simulated interrupt.  On
    return, the client must build a callback info structure and store
    the returned ES:BX in the "next" field, then return the address of
    its own callback info structure.
  a client program must add itself to the notification chain if it
    provides services to other programs; before terminating, it must
    remove itself from the chain by calling the task switcher's entry
    point with AX=0005h (see #02819)
  the task switcher entry point should not be saved, as it is subject to
    change and will be provided on any notification call
```

```
  the Windows 3.1 Standard Mode supports this API
SeeAlso: AX=160Bh,AX=4B02h

Format of task switcher callback info structure:
Offset  Size  Description (Table 02817)
 00h  DWORD pointer to next callback info structure
 04h  DWORD pointer to notification function (see #02818)
 08h  DWORD reserved
 0Ch  DWORD address of zero-terminated list of API info structures
    (see #02821)

(Table 02818)
Values task switcher notification function is called with:
  AX = function
      0000h switcher initialization
    Return: AX = status
          0000h if OK to load
          nonzero to abort task switcher
      0001h query suspend
    BX = session ID
    Return: AX = status
          0000h if OK to switch session
          0001h if not
      0002h suspend session
    BX = session ID
    interrupts disabled
    Return: AX = 0000h if OK to switch session
         = 0001h if not
      0003h activate session
    BX = session ID
    CX = session status flags
      bit 0: set if first activation of session
      bits 1-15: reserved (0)
    interrupts disabled
    Return: AX = 0000h
      0004h session active
    BX = session ID
    CX = session status flags
      bit 0: set if first activation of session
      bits 1-15: reserved (0)
    Return: AX = 0000h
```

```
      0005h create session
    BX = session ID
    Return: AX = 0000h if OK to create session
         = 0001h if not
      0006h destroy session
    BX = session ID
    Return: AX = 0000h
      0007h switcher termination
    BX = flags
        bit 0: set if calling switcher is only switcher loaded
        bits 1-15: reserved (0)
    Return: AX = 0000h
  ES:DI -> task switcher entry point (see #02819)
Notes:  function 0000h is generally called by the program which controls or
    invokes the task switcher, rather than by the task switcher itself;
    the entry point supplied to this function is not necessarily the
    entry point to the task switcher itself, and may be 0000h:0000h.  If
    any client indicates that loading is not possible, all clients will
    be called with function 0007h; thus it is possible for a client to
    receive a termination notice without a corresponding initialization
    notice.
  except for functions 0002h and 0003h, the notification handler is
    called with interrupts enabled and may make any INT 21h function
    call; interrupts must not be enabled in functions 0002h and 0003h
  function 0007h may be called with ES:DI = 0000h:0000h if the entry
    point is no longer valid
--------T-2F4B02BX0000-----------------------
INT 2F - DOS 5+ TASK SWITCHER - INSTALLATION CHECK
  AX = 4B02h
  BX = 0000h
  ES:DI = 0000h:0000h
Return: ES:DI = 0000h:0000h if task switcher not loaded
  ES:DI -> task switcher entry point (see #02819) if loaded
      AX = 0000h
Notes:  the returned entry point is that for the most-recently loaded task
    switcher; the entry points for prior task switchers may be determined
    with the "get version" call (see #02819)
  this function is supported by PC Tools v8+ CPTASK
SeeAlso: AX=4A05h,AX=4B03h

(Table 02819)
```

```
Call task switcher entry point with:
  AX = 0000h get version
    Return: CF clear if successful
          AX = 0000h
          ES:BX -> task switcher version struct (see #02820)
      CF set if unsupported function
  AX = 0001h test memory region
    ES:DI -> first byte to be tested
    CX = size of region to test
    Return: CF clear if successful
          AX = memory type of tested region
        0000h global
        0001h global and local
        0002h local (replaced on session switch)
      CF set if unsupported function
  AX = 0002h suspend switcher
    ES:DI -> new task switcher's entry point
    Return: CF clear if successful
          AX = state
        0000h switcher has been suspended
        0001h switcher not suspended, new switcher must
          abort
        0002h switcher not suspended, but new switcher
          may run anyway
      CF set if unsupported function
  AX = 0003h resume switcher
    ES:DI -> new task switcher's entry point
    Return: CF clear if successful
          AX = 0000h
      CF set if unsupported function
  AX = 0004h hook notification chain
    ES:DI -> callback info structure to be added to chain
        (see #02817)
    Return: CF clear if successful
          AX = 0000h
      CF set if unsupported function
  AX = 0005h unhook notification chain
    ES:DI -> callback info structure to be removed from chain
        (see #02817)
    Return: CF clear if successful
          AX = 0000h
```

```
      CF set if unsupported function
  AX = 0006h query API support
    BX = asynchronous API identifier
    Return: CF clear if successful
          AX = 0000h
          ES:BX -> API info structure (see #02821) for the
              client which provides the highest
              level of API support
      CF set if unsupported function

Format of task switcher version structure:
Offset  Size  Description (Table 02820)
 00h  WORD  major version of supported protocol  (current protocol is 1.0)
 02h  WORD  minor version of supported protocol
 04h  WORD  major version of task switcher
 06h  WORD  minor version of task switcher
 08h  WORD  task switcher ID (see AX=4B03h)
 0Ah  WORD  operation flags
    bit 0: set if task switcher disabled
    bits 1-15: reserved (0)
 0Ch  DWORD pointer to ASCIZ task switcher name
    ("MS-DOS Shell Task Switcher" for DOSSHELL task switcher)
 10h  DWORD pointer to previous task switcher's entry point or 0000h:0000h

Format of API info structure:
Offset  Size  Description (Table 02821)
 00h  WORD  size of structure in bytes (000Ah)
 02h  WORD  API identifier
    0001h NetBIOS
    0002h 802.2
    0003h TCP/IP
    0004h LAN Manager named pipes
    0005h Novell NetWare IPX
 04h  WORD  major version \ of highest version of API for which the support
 06h  WORD  minor version / level specified in the next field is provided
 08h  WORD  support level
    0001h minimal support
    0002h API-level support
    0003h switcher compatibility
    0004h seamless compatibility
--------T-2F4B03-----------------------------
```

```
INT 2F - DOS 5+ TASK SWITCHER - ALLOCATE SWITCHER ID
  AX = 4B03h
  ES:DI -> task switcher entry point (see #02819)
Return: AX = 0000h
  BX = switcher ID (0001h-000Fh), or 0000h if no more available
Notes:  if a task switcher has determined that it is the first to be loaded, it
    must allocate an identifier for itself and provide this function to
    all subsequent task switchers; if it is not the first to be loaded,
    it must call this function to allocate an ID.  The switcher ID is
    used as the high four bits of all session identifiers to ensure
    unique session IDs.
  if no more switcher IDs are available, the new task switcher making the
    call must terminate or disable itself
  the task switcher providing the identifiers may call the new task
    switcher's entry point as needed
  this call is available from within DOSSHELL even if the task switcher
    is not installed
  this function is supported by PC Tools v8+ CPTASK, but appears to
    always return an ID of 0000h
SeeAlso: AX=4B02h,AX=4B04h
--------T-2F4B04-----------------------------
INT 2F - DOS 5+ TASK SWITCHER - FREE SWITCHER ID
  AX = 4B04h
  BX = switcher ID
  ES:DI -> task switcher entry point (see #02819)
Return: AX = 0000h
  BX = status
      0000h successful
      other error (invalid ID or ID not allocated)
Notes:  called by a task switcher when it exits, unless it was the first loaded
    and is providing the support for AX=4B03h and AX=4B04h
  the task switcher providing the identifiers may call the terminating
    task switcher's entry point as needed
  this call is available from within DOSSHELL even if the task switcher
    is not installed
  this call is supported by PC Tools v8+ CPTASK, but appears to return
    sucessfully no matter which ID is given
SeeAlso: AX=4B02h,AX=4B03h
--------T-2F4B05-----------------------------
INT 2F C - DOS 5+ TASK SWITCHER - IDENTIFY INSTANCE DATA
  AX = 4B05h
```

```
  ES:BX = 0000h:0000h
  CX:DX -> task switcher entry point (see #02819)
Return: ES:BX -> startup info structure (see #02822) or 0000h:0000h
Notes:  called by task switcher
  clients with instance data should hook this call, pass it through to
    the previous handler with unchanged registers using a simulated
    interrupt.  On return, the client should create a startup info
    structure (see #02822), store the returned ES:BX in the "next"
    field, and return the address of the created structure in ES:BX
  all MS-DOS function calls are available from within this call
  this function is supported by Novell DOS 7 DOSKEY, with structure v3.00
SeeAlso: AX=1605h,AX=160Bh,AX=4B02h

Format of task switcher startup info structure:
Offset  Size  Description (Table 02822)
 00h  2 BYTEs major, minor version of info structure (03h,00h)
 02h  DWORD pointer to next startup info structure or 0000h:0000h
 06h  DWORD 0000h:0000h (ignored)
 0Ah  DWORD ignored
 0Eh  DWORD pointer to instance data records (see #02823)

Format of one instance data record in array:
Offset  Size  Description (Table 02823)
 00h  DWORD address of instance data (end of array if 0000h:0000h)
 04h  WORD  size of instance data
--------W-2F4B06-----------------------------
INT 2F - MS Windows - WIN.COM - GET ??? POINTER TO WIN.COM
  AX = 4B06h
Return: AX = 0000h
  ES:BX -> ??? function in WIN.COM
Note: the entry point is called with
      AX = 0001h or 0003h
      BX = ???
SeeAlso: AX=4B80h
--------W-2F4B20-----------------------------
INT 2F - MS Windows 3+ - WIN.COM - SET PROGRAM TO EXECUTE ON EXIT
  AX = 4B20h
Return: AX = 0000h if successful
      DX:CX -> 256-byte buffer for pathname and commandline (see #02824)
Notes:  when the Windows function ExitWindows is called with an exit code of
    44h, WIN.COM executes the program specified in the returned buffer
```

```
    and then restarts Windows
  the returned address is a real-mode segment:offset value
SeeAlso: AX=4B21h

Format of WIN.COM buffer:
Offset  Size  Description (Table 02824)
 00h 128 BYTEs  commandline for program (count byte, command tail, 0Dh)
 80h 128 BYTEs  ASCIZ pathname of program to execute
Note: the order above is for a Windows95 DOS box; it may be reversed under
    Windows 3.x
--------W-2F4B21-----------------------------
INT 2F - Windows95 - WIN.COM - GET NESTING LEVEL
  AX = 4B21h
Return: AH = 00h if WIN.COM already active
      AL = number of instances of WIN.COM in memory
SeeAlso: AX=4B20h
--------K-2F4B52-----------------------------
INT 2F - KeyRus v7.3 - API
  AX = 4B52h ('KR')
  BL = function number
      00h installation check
    Return: AL = 82h if installed
          BH = major version number
          BL = minor version number
          ES destroyed
      01h get driver status
    Return: AL??? = current status (see #02825)
      02h set driver state
    AL = new driver state (see #02825)
      03h blank screen (if blanking enabled when TSR was loaded)
      04h unblank screen
      4Ch switch to English keyboard mode
      90h switch to Russian keyboard mode
Return:

Bitfields for KeyRus driver status:
Bit(s)  Description (Table 02825)
 1-0  language mode
  00 Latin
  01 Russian
  10 Alternative
```

```
  11 unused
 2  allow character loading (if disabled, use ROM fonts)
 3  English keyboard support
 7-4  used internally (read-only)
--------W-2F4B80-----------------------------
INT 2F - MS Windows - WSWAP.EXE - RESET INTERNAL VARIABLES
  AX = 4B80h
Return: nothing
Note: called by WINOLDAP.MOD
SeeAlso: AX=4B06h
--------p-2F4C-------------------------------
INT 2F U - Advanced Power Management
  AH = 4Ch
  AL = function
       00h version check
       01h suspend system requested
       FFh suspend/resume battery notification
  ???
Return: ???
----------2F4D-------------------------------
INT 2F U - KKCFUNC
  AH = 4Dh
  AL = function
       00h get function address
       01h get error number of last call to KKC DOS function
       02h register/release KKC
       03h get table address
  !!! details to follow
Return: ???
Notes:  This API is provided by KKCFUNC.SYS, a support driver for Kana Kanji
    Converters (KKC), which is used to handle multiple client KKCs and
    provide all the necessary framework to call DOS functions at any
    time.
  also called by MSKK
  For AL > 3, KKCFUNC passes the call down to the original INT 2Fh,
    as recorded at initialization.
--------N-2F4E53BL00-------------------------
INT 2F - SilverNET v2+ - INSTALLATION CHECK
  AX = 4E53h ("NS")
  BL = 00h (function "installation check")
  BH = module ID (see #02826)
```

```
Return: AX = 0000h if specified module installed
  BX = 4E53h if installed
Program: SilverNET is an SMB-compatible peer-to-peer NOS for DOS or
    Windows systems, by Net-Source, Inc. of Santa Clara, CA.
SeeAlso: AX=4E53h/BL=01h,AX=4E53h/BL=02h,AX=B800h,AX=B809h

(Table 02826)
Values for SilverNET module ID:
 01h  SilverCACHE
 02h  Workstation
 03h  NetBIOS
 04h  Peer
 20h  NS Share
 80h  NetWare help TSR
--------N-2F4E53BL01-------------------------
INT 2F - SilverNET - GET RUNTIME PARAMETER
  AX = 4E53h ("NS")
  BL = 01h (function "get runtime parameter")
  BH = module ID (see #02826)
  CX = parameter index (see #02827,#02829,#02830)
Return: AX = WORD value at specified index (see #02828)
Desc: retrieve a word of data from the specified SilverNET module

(Table 02827)
Values for SilverNET Peer parameter index (* = read-only):
 00h *  maximum outstanding SMB buffers
 02h *  maximum logged-in nodes
 04h *  number of shareable resources
 06h *  number of characters to print per time slice
 08h *  number of printers that can be shared
 0Ah *  number of nodes logged in
 0Ch *  number of files to allow opened
 0Eh  how fast to despool (/PSLICE)
 10h  audit flag
 24h *  far pointer to resource table (each resource is 96 bytes in length)
 32h *  far pointer to SFT (internal if SilverNET files > CONFIG.SYS files,
  else DOS SFT)
 36h  spool flags (see #02828)
SeeAlso: #02829

Bitfields for spool flags:
```

```
Bit(s)  Description (Table 02828)
 0  LPT1 needs despooling
 1  LPT2 needs despooling
 2  LPT2 needs despooling
 4  COM1 needs despooling
 5  COM2 needs despooling
 6  COM3 needs despooling
SeeAlso: #02827

(Table 02829)
Values for NS Share parameter index (* = read-only):
 00h  version number (high byte = minor, low byte = major)
 10h *  segment of first lock record (other records in consecutive paragraphs)
  (if PSP field = 0000h, lock record is free)
 12h *  maximum possible number of lock records
 14h *  starting segment of sharing buffer
  (NS Share's sharing records are identical to DOS SHARE except that
    fields which are normally offsets into SHARE are segment numbers)
 18h *  size of sharing buffer in paragraphs
 1Ah *  total free paragraphs in sharing buffer
 1Ch *  current number of shared files
 1Eh *  current number of locked records
SeeAlso: #02827,#02830

(Table 02830)
Values for Workstation parameter index (* = read-only):
 00h  version number (high byte = minor, low byte = major)
 02h *  size of each network buffer for file operations
 04h *  number of redirector file buffers
 06h *  size of each print cache buffer
 08h *  number of network LPT printers
 0Ch  flush time in ticks (idle time on network printer before flushing)
 0Eh  (16 WORDs) last active time for each printer
 2Eh *  stub segment if program split into two parts
 60h  receive name number for datagram listens
 62h *  18-byte machine name
 74h *  LASTDRIVE (01h = A:, etc.)
 7Ch  row number of message box on screen
 7Eh  message time in clock ticks
 82h *  number of network adapters in use
 84h  station ID broadcast flag (never set on redirectors)
```

```
 96h *  NetBIOS names left
 98h *  NCBs left
 9Ah *  sessions left
 A2h *  total number of network printers (LPT+COM)
 A4h *  number of serial network printers
 A8h *  segment containing file cache buffers
 AAh *  segment containing print cache buffers
 ACh *  bytes remaining free in HMA before program loaded
 AEh *  start of free memory in HMA
 B2h *  flag: using HMA
SeeAlso: #02829
--------N-2F4E53BL02-------------------------
INT 2F - SilverNET - SET RUNTIME PARAMETERS
  AX = 4E53h ("NS")
  BL = 02h (function "set runtime parameters")
  BH = module ID (see #02826)
  CX = parameter index (see #02829,#02830)
  DX = new value for specified parameter
Desc: set a WORD value in the specified SilverNET module
Note: not all indexed parameters are writable; modifying a read-only
    parameter can result in system crashes
SeeAlso: AX=4E53h/BL=00h,AX=4E53h/BL=01h
--------N-2F5100-----------------------------
INT 2F U - ODIHLP.EXE - INSTALLATION CHECK
  AX = 5100h
Return: AL = FFh if installed
      BX = 0000h
      DX:SI -> signature string "ODI$HLP$"
Program: ODIHLP is a real-mode helper allowing the Windows for Workgroups 3.11
    protected-mode NDIS3 protocol to work with real-mode ODI drivers
    and LSL.COM
Note: the returned signature string might be the first field of a structure
SeeAlso: AX=C000h"LSL.COM"
--------k-2F5200-----------------------------
INT 2F - JAM.SYS v1.10+ - "GetVersion" - INSTALLATION CHECK
  AX = 5200h
Return: AH = 80h (successful) if installed
  BX = internal JAM.SYS version number
  CX = size of JAMINFO structure (see #02831,#02832)
  DX = JAM.SYS segment address
Program: JAM.SYS is a main component of the JAM Real-Time Data Compression
```

```
    Utilities by George A. Reznik and friends (JAM Software).
SeeAlso: AX=5201h

Format of JAMINFO v1.10 structure:
Offset  Size  Description (Table 02831)
 00h 25 BYTEs extended BIOS parameter block (BPB)
 19h 11 BYTEs ???
 25h  DWORD total number of sectors in JAM archive file
    (size of compressed data area)
 29h  BYTE  flags (see #02833)
 2Ah 127 BYTEs  full JAM archive file name
 A9h  WORD  the number of fragments in archive file
 ABh 96 BYTEs archive file fragmentation list -
    array of 16 FRAGMENT structures (see #02834)
10Bh  DWORD address of the host-drive DPB (Drive Parameter Block)
10Fh  DWORD number of free sectors in JAM archive file
113h  WORD  device status word (see #02835)
SeeAlso: #02832

Format of JAMINFO v1.20 structure:
Offset  Size  Description (Table 02832)
 00h 25 BYTEs extended BIOS parameter block (BPB)
 19h  BYTE  physical driver number
 1Ah  BYTE  reserved
 1Bh  BYTE  extended boot record signature
 1Ch  DWORD volume serial number
 20h 11 BYTEs volume label
 2Bh  8 BYTEs file system ID
 33h  DWORD total number of sectors in JAM archive file
    (size of compressed data area)
 37h  BYTE  flags (see #02833)
 38h 128 BYTEs  full JAM archive file name
 B8h  WORD  the number of fragments in archive file
 BAh 96 BYTEs archive file fragmentation list -
    array of 16 FRAGMENT structures (see #02834)
11Ah  DWORD address of the host-drive DPB (Drive Parameter Block)
11Eh  DWORD number of free sectors in JAM archive file
122h  WORD  device status word (see #02835)
Note: the first 33h bytes are copied from the archive file's boot sector
SeeAlso: #02831
```

```
Bitfields for JAMINFO flags:
Bit(s)  Description (Table 02833)
 2-0  reserved
 3  (v1.20+)
 4  enable direct write requests (Int 26h, non-DOS requests, etc.)
 5  read-only mode
 6  no write-behind caching
 7  full undelete-compatible allocation strategy
SeeAlso: #02831,#02832

Format of JAM FRAGMENT structure:
Offset  Size  Description (Table 02834)
 00h  WORD  starting sector (low word)
 02h  BYTE  starting sector (high byte)
 03h  WORD  size of fragment (low word)
 05h  BYTE  size of fragment (high byte)
SeeAlso: #02831,#02832

(Table 02835)
Values for JAM.SYS status (high byte):
 00h  successful
 01h  drive is not a JAM drive
 02h  drive is already attached
 03h  archive file cluster size value is larger than driver's one
 04h  drive is not attached
 05h  drive is locked
 06h  drive is not locked
 07h  bad physical-level request
 08h  host drive reading/writing error
 09h  bad entries in JAM descriptor table
 0Ah  compressed data integrity error
 0Bh  archive file overflow
 0Ch  bad DOS request
 0Dh  incorrect parameters in JAMINFO structure
Note: the low byte of the status is the DOS error code for the Host drive
SeeAlso: #02598 at INT 2F/AX=0802h
--------k-2F5201-----------------------------
INT 2F - JAM.SYS v1.10+ - "GetInfo" - GET COMPRESSED DRIVE INFORMATION
  AX = 5201h
  DL = compressed drive number (0-default, 1-A:, etc.)
  DS:BX -> buffer for JAMINFO structure (see #02831,#02832)
```

```
Return: AH = status (00h,01h) (see #02835)
SeeAlso: AX=5200h
--------k-2F5202-----------------------------
INT 2F - JAM.SYS v1.10+ - "Attach" - MOUNT COMPRESSED DRIVE
  AX = 5202h
  DL = drive number (0-default, 1-A:, etc.) to attach to the JAM
       archive file
  DS:BX -> pointer to JAMINFO structure (see #02831,#02832), which
       contains parameters of the JAM file to mount, and pointer
       to the host drive DPB (i.e. DPB of the drive on which the
       JAM file is located)
Return: AH = status (00h,02h,03h,08h,09h,0Dh) (see also #02835)
      03h archive file cluster size value is larger than driver's - not
      mounted
      09h bad entries in JAM descriptor table - file mounted read-only
  AL = host drive error code (see #02598 at INT 2F/AX=0802h)
SeeAlso: AX=5203h
--------k-2F5203-----------------------------
INT 2F - JAM.SYS v1.10+ - "Detach" - UNMOUNT COMPRESSED DRIVE
  AX = 5203h
  DL = drive number (0-default, 1-A:, etc.) to detach
Return: AH = status (00h,01h,04h,05h,08h,09h,0Bh,0Dh) (see #02835)
  AL = host drive error code (see #02598 at INT 2F/AX=0802h)
SeeAlso: AX=5202h
--------k-2F5204-----------------------------
INT 2F - JAM.SYS v1.10+ - "Lock" - LOCK COMPRESSED DRIVE
  AX = 5204h
  DL = drive number (0-default, 1-A:, etc.) to lock
Return: AH = status (00h,01h,04h,05h,08h,09h,0Bh,0Dh) (see #02835)
  AL = host drive error code (see #02598 at INT 2F/AX=0802h)
SeeAlso: AX=5205h, AX=5206h, AX=5207h
--------k-2F5205-----------------------------
INT 2F - JAM.SYS v1.10+ - "UnLock" - UNLOCK COMPRESSED DRIVE
  AX = 5205h
  DL = drive number (0-default, 1-A:, etc.) to unlock
Return: AH = status (00h,01h,04h,06h,08h,09h,0Dh) (see #02835)
  AL = host drive error code (see #02598 at INT 2F/AX=0802h)
SeeAlso: AX=5204h, AX=5206h, AX=5207h
Note: Lock and UnLock functions were added to the JAM API to prevent
    asynchronous physical-level access (see AX=5206h,AX=5207h) to
    compressed data on JAM drives. In other words, two or more programs
```

```
    which use JAM API (say, JMAX optimizer and JCHKDSK - disk checker)
    cannot be run on the same JAM drive simultaneously.
--------k-2F5206-----------------------------
INT 2F - JAM.SYS v1.10+ - "Read" - PHYSICAL READ DATA FROM JAM ARCHIVE
  AX = 5206h
  DL = drive number (0-default, 1-A:, etc.)
  DS:BX -> disk transfer packet (see #02836)
Return: AH = status (00h,01h,04h,06h,07h,08h,0Dh) (see #02835)
  AL = host drive error code (see #02598 at INT 2F/AX=0802h)
Program: JAM.SYS is a main component of the JAM Real-Time Data Compression
    Utilities by George A. Reznik and friends (JAM Software).
SeeAlso: AX=5207h

Format of disk transfer packet:
Offset  Size  Description (Table 02836)
 00h  DWORD sector number
 04h  WORD  number of sectors to read(write)
 06h  DWORD transfer address
--------k-2F5207-----------------------------
INT 2F - JAM.SYS v1.10+ - "Write" - PHYSICAL WRITE DATA TO JAM ARCHIVE
  AX = 5207h
  DL = drive number (0-default, 1-A:, etc.)
  DS:BX -> disk transfer packet (see #02836)
Return: AH = status (00h,01h,04h,06h,07h,08h,0Dh) (see #02835)
  AL = host drive error code (see #02598 at INT 2F/AX=0802h)
SeeAlso: AX=5206h
--------p-2F53-------------------------------
INT 2F U - POWER.EXE - APM event broadcasting???
  AH = 53h
  AL = event???
       05h CPU idle
       0Bh PM event broadcast API
Return: ???
Note: called by MS Windows 3.1 POWER.DRV; hooked by MS Mouse driver v8.20+
    and PC-Cache v8.0
SeeAlso: AX=530Bh,AX=5400h,INT 33/AX=002Fh
--------p-2F530B-----------------------------
INT 2F U - ??? (MOUSEPWR.COM, others) - ???
  AX = 530Bh
  BX = subfunction
       0003h ???
```

```
      0004h ???
  ???
Return: ???
Note: it appears that subfunction 0003h reads or restores the current mouse
    settings (the MS Mouse driver hooks AX=530Bh), and 0004h might be
    the converse
--------p-2F5400-----------------------------
INT 2F U - POWER.EXE - INSTALLATION CHECK
  AX = 5400h
Return: AX = POWER.EXE version (AH = major, AL = minor) if installed
  BX = 504Dh ("PM")
  CF clear
Note: called by MS Windows 3.1 POWER.DRV
SeeAlso: AH=53h,AX=5401h,AX=5402h,AX=5481h,AX=5482h
--------p-2F5401-----------------------------
INT 2F U - POWER.EXE - GET/SET POWER STATUS
  AX = 5401h
  BH = function
      00h get status
    Return: BL = current power management status (see #02837)
      01h set status
    BL = new power managment status (see #02837)
Return: AX = function status (see #02838)
Note: called by MS Windows 3.1 POWER.DRV
SeeAlso: AH=53h,AX=5400h,AX=5402h,AX=5403h

Bitfields for power management status:
Bit(s)  Description (Table 02837)
 0  POWER.EXE power management enabled
 1  APM firmware power management enabled
 2-7  reserved (0)
Notes:  bit 1 is ignored if there is no APM firmware
  bits 1-0: 00 = POWER OFF, 10 = POWER STD, 11 = POWER ADV

(Table 02838)
Values for POWER.EXE function status:
 0000h  successful
 0002h  "ERROR_PM_ALREADY_CONNECTED"
 0003h  "ERROR_PM_NOT_CONNECTED"
 0087h  "ERROR_PM_INVALID_PARAMETER"
--------p-2F5402-----------------------------
```

```
INT 2F U - POWER.EXE - GET/SET IDLE DETECTION STRATEGY
  AX = 5402h
  BH = subfunction
       00h get
       other set
    BL = detection strategy (00h-0Fh or FFh)
Return: BX = current/new detection strategy
SeeAlso: AH=53h,AX=5400h,AX=5401h,AX=5481h,AX=5482h
--------p-2F5403-----------------------------
INT 2F U - POWER.EXE - GET/SET ADVANCED POWER MANAGEMENT SETTING
  AX = 5403h
  BX = new power management setting or 0000h to get current setting
Return: AX = status
      0000h successful
    BX = power management setting (see #02839)
      other error code
SeeAlso: AX=5401h,AX=5480h

(Table 02839)
Values for power management setting:
 0001h-0005h "min"
 0006h       "reg"
 0007h-0008h "max"
--------t-2F5453-----------------------------
INT 2F - TesSeRact RAM-RESIDENT PROGRAM INTERFACE
  AX = 5453h
  BX = subfunction
       00h installation check
    CX = 0000h
    DS:SI -> 8-char blank-padded name (see #02840)
    Return: AX = FFFFh installed
        CX = ID number of already-installed copy
      AX = anything else, not installed
        CX = ID number for TSR when installed
      01h get user parameters
    CX = TSR ID number
    Return: AX = status
          0000h successful
        ES:BX -> user parameter block (see #02841)
          nonzero failed
      02h check if hotkey in use
```

```
CL = scan code of hot key (see #00006)
Return: AX = FFFFh hot key conflicts with another TSR
       otherwise safe to use the hotkey
  03h replace default critical error handler
CX = TSR ID number
DS:SI -> new routine for INT 24h
Return: AX = nonzero, unable to install new handler
  04h get internal data area
CX = TSR ID number
Return: AX = status
      0000h successful
    ES:BX -> TSR's internal data area (see #02842)
      nonzero, TSR not found
  05h set multiple hot keys
CX = TSR ID number
DL = number of additional hot keys to allocate
DS:SI -> table of hot keys
  BYTE  hotkey scan code (see #00006)
  BYTE  hotkey shift state
  BYTE  flag value to pass to TSR (nonzero)
Return: AX = nonzero, unable to install hot keys
  06h - 0Fh reserved
  10h enable TSR
CX = TSR ID number
Return: AX = nonzero, unable to enable
  11h disable TSR
CX = TSR ID number
Return: AX = nonzero, unable to disable
  12h unload TSR
CX = TSR ID number
Return: AX = nonzero, invalid TSR number
Note: if any interrupts used by TSR have been grabbed by
  another TSR, the TesSeRact routines will wait until
  it is safe to remove the indicated TSR from memory
  13h restart TSR
CX = TSR ID number of TSR which was unloaded but is still in
     memory
Return: AX = nonzero, unable to restart TSR
  14h get status word
CX = TSR ID number
Return: AX = FFFFh invalid ID number
```

```
        = other, successful
       BX = bit flags
    15h set status word
  CX = TSR ID number
  DX = new bit flags
  Return: AX = nonzero, unable to set status word
    16h get INDOS state at popup
  CX = TSR ID number
  Return: AX = 0000h successful
        BX = value of INDOS flag
    17h - 1Fh reserved
    20h call user procedure
  CX = TSR ID number
  ES:DI -> user-defined data
  Return: AX = 0000h successful
    21h stuff keystrokes into keyboard buffer
  CX = TSR ID number
  DL = speed
      00h stuff keystrokes only when buffer is empty
      01h stuff up to four keystrokes per clock tick
      02h stuff up to 15 keystrokes per clock tick
  DH = scan code flag
      if zero, buffer contains alternating ASCII and scan codes
      if nonzero, buffer contains only ASCII codes
  SI = number of keystrokes
  ES:DI -> buffer to stuff
  Return: AX = 0000h success
         F0F0h user aborted with ^C or ^Break
         other unable to stuff keystrokes
    22h (v1.10) trigger popup
  CX = TSR ID number
  Return: AX = 0000h success, TSR will either pop up or beep to
         indicate that it is unable to pop up
         nonzero invalid ID number
    23h (v1.10) invoke TSR's background function
  CX = TSR ID number
  Return: AX = 0000h success
         FFFFh not safe to call background function
         nonzero invalid ID number
    24h - 2Fh reserved
Notes:  Borland's THELP.COM popup help system for Turbo Pascal and Turbo C
```

```
    (versions 1.x and 2.x only) fully supports the TesSeRact API, as
    do the SWAP?? programs by Innovative Data Concepts.
  AVATAR.SYS supports functions 00h and 01h (only the first three fields
    of the user parameter block) using the name "AVATAR  "
SeeAlso: AX=CAFEh,INT 16/AX=55FFh,INT 2D"AMIS"
Index:  installation check;TesSeRact TSR interface|uninstall;TesSeRact

(Table 02840)
Values for TesSeRact names:
 "AVATAR  " AVATAR.SYS
 "QeditTSR" TSR version of SemWare's Qedit editor
 "SCRNBLNK" Trusted Access screen blanker

Format of TesSeRact User Parameter Block:
Offset  Size  Description (Table 02841)
 00h  8 BYTEs blank-padded TSR name
 08h  WORD  TSR ID number
 0Ah  DWORD bitmap of supported functions
 0Eh  BYTE  scan code of primary hotkey (see #00006)
    00h = pop up when shift states match
    FFh = no popup (if shift state also FFh)
 0Fh  BYTE  shift state of primary hotkey
    FFh = no popup (if scan code also FFh)
 10h  BYTE  number of secondary hotkeys
 11h  DWORD pointer to extra hotkeys set by func 05h
 15h  WORD  current TSR status flags
 17h  WORD  PSP segment of TSR
 19h  DWORD DTA for TSR
 1Dh  WORD  default DS for TSR
 1Fh  DWORD stack at popup
 23h  DWORD stack at background invocation
Index:  hotkeys;TesSeRact TSR interface

Format of TSR internal data area:
Offset  Size  Description (Table 02842)
 00h  BYTE  revision level of TesSeRact library
 01h  BYTE  type of popup in effect
 02h  BYTE  INT 08 occurred since last invocation
 03h  BYTE  INT 13 occurred since last invocation
 04h  BYTE  active interrupts
 05h  BYTE  active soft interrupts
```

```
 06h  BYTE  DOS major version
 07h  BYTE  how long to wait before popping up
 08h  DWORD pointer to INDOS flag
 0CH  DWORD pointer to DOS critical error flag
 10h  WORD  PSP segment of interrupted program
 12h  WORD  PSP segment of prog interrupted by INT 28
 14h  DWORD DTA of interrupted program
 18h  DWORD DTA of program interrupted by INT 28
 1Ch  WORD  SS of interrupted program
 1Eh  WORD  SP of interrupted program
 20h  WORD  SS of program interrupted by INT 28
 22h  WORD  SP of program interrupted by INT 28
 24h  DWORD INT 24 of interrupted program
 28h  3 WORDs DOS 3.0+ extended error info
 2Eh  BYTE  old BREAK setting
 2Fh  BYTE  old VERIFY setting
 30h  BYTE  were running MS WORD 4.0 before popup
 31h  BYTE  MS WORD 4.0 special popup flag
 32h  BYTE  enhanced keyboard call in use
 33h  BYTE  delay for MS WORD 4.0
11 times (for INTs 08h,09h,13h,16h,1Ch,21h,28h,2Fh,1Bh,23h, and 24h):
  DWORD old interrupt vector
  BYTE  interrupt number
  WORD  offset in TesSeRact code segment of new interrupt handler
--------p-2F5480-----------------------------
INT 2F U - POWER.EXE - GET/SET ???
  AX = 5480h
  BX = direction
       0000h get
       other set
  CX = size of buffer (at least 0010h)
  DS:SI -> buffer
Return: AX = status
       0000h successful
       other error code
SeeAlso: AX=5400h,AX=5481h,AX=548Fh
--------p-2F5481-----------------------------
INT 2F U - POWER.EXE - GET STATISTICS
  AX = 5481h
  BX = which statistics
       0000h idle detection
```

```
      0001h APM statistics
  CX = length of buffer in bytes
  DS:SI -> buffer for statistics (see #02843,#02844)
Return: AX = status
      0000h successful
      0071h "ERROR_PM_BUFFER_TOO_SMALL"
      0087h "ERROR_PM_INVALID_PARAMETER"
SeeAlso: AH=53h,AX=5400h,AX=5480h,AX=5402h,AX=5482h

Format of POWER.EXE idle detection statistics:
Offset  Size  Description (Table 02843)
 00h  DWORD "CPU_ON_TIME" total time CPU is active with POWER.EXE idle
      detection enabled, in timer ticks
 04h  DWORD "CPU_IDLE_TIME" timer ticks during which CPU was idle
    (divide by previous to get idle rate)
 08h  DWORD total idle calls
 0Ch  DWORD "TOTAL_APP_IDLE" total INT 2Fh idle calls
 10h  DWORD "TOTAL_DOS_YIELD" total INT 28h idle calls
 14h  DWORD "TOTAL_KEY_IDLE" total INT 16h idle calls
 18h  DWORD "TOTAL_DOS_IDLE" total INT 2Ah idle calls

Format of APM statistics:
Offset  Size  Description (Table 02844)
 00h  DWORD "RESUME_COUNT" total number of resumes since last APM_ENABLE
--------p-2F5482-----------------------------
INT 2F U - POWER.EXE - GET/SET APM POLLING FREQUENCY
  AX = 5482h
  BX = new polling frequency or 0000h to get current frequency
Return: AX = 0000h (successful)
  BX = current frequency if BX=0000h on entry
SeeAlso: AH=53h,AX=5400h,AX=5401h,AX=5480h,AX=5481h,AX=548Fh
--------p-2F548F-----------------------------
INT 2F U - POWER.EXE - GET/SET ???
  AX = 548Fh
  BX = ??? or 0000h to get current ???
Return: AX = 0000h (successful)
  BX = current ???
  CX = ???
SeeAlso: AX=5400h,AX=5480h,AX=5482h
--------l-2F5500-----------------------------
INT 2F U - DOS 5+ - COMMAND.COM INTERFACE
```

```
  AX = 5500h
Return: AX = 0000h if an instance of COMMAND.COM is already running
  DS:SI -> entry point table
Notes:  used to access the shareable portion of COMMAND.COM, which may have
    been moved into the HMA; only the primary COMMAND.COM retains this
    portion
  procedures called from a dispatcher in COMMAND's resident portion;
    most assume that the segment address of the resident portion is on
    the stack and are thus not of general use
  DR PalmDOS up to DR-DOS 7.03 COMMAND.COM do not support this call.
  When loading the default command processor (no SHELL= directive in
    CONFIG.SYS), MS-DOS 6.0-6.22 IO.SYS & PC DOS 6.1-2000 IBMBIO.COM
    check a signature (see #04099) in the COMMAND.COM file image to test
    if it is actually their own shell and has the correct version, before
    they will launch it. If the signature is not found, the message
    "Invalid COMMAND.COM" will be displayed. This test seems to have
    vanished with MS-DOS 7+, as it uses an .EXE style file format.
  In addition to this, the MS-DOS/PC DOS COMMAND.COM checks the version
    of the underlaying OS to see if it is their own. Hence, the
    PC DOS 6.1 (and by SETVER version faking also the PC DOS 7 and 2000)
    COMMAND.COM also run on Novell DOS 7 - DR DOS 7.03, which all
    identify themselves as DOS API level 6.0 and IBM OEM (not yet tested
    as primary shell).  PC DOS 5.0 COMMAND.COM should run on DR PalmDOS
    (untested).
  The MS-DOS 7+ COMMAND.COM seems to no longer perform this kind of
    version check any more, which can cause a serious deadlock situation
    on a multi-boot system with DR-DOS installed, when the MS-DOS 7+
    COMMAND.COM is placed in C:\. When IBMBIO.COM attempts to load the
    MS-DOS 7+ COMMAND.COM as a primary shell under DR-OpenDOS 7.02 to
    DR-DOS 7.03 no error message will be displayed by COMMAND.COM, but
    the machine will just hang. To defuse this situation, DR-DOS 7.02+
    IBMBIO.COM was changed to still scan for a SHELL= directive in
    [D]CONFIG.SYS even in F5-mode. For maximum safety, the MS-DOS 7
    COMMAND.COM should be moved from C:\ to C:\WINDOWS\COMMAND\
    and a DCONFIG.SYS file should be created containing for example
    SHELL=C:\DRDOS\COMMAND.COM instead. If a MS-DOS CONFIG.SYS file
    exists, it should contain a SHELL=C:\WINDOWS\COMMAND\COMMAND.COM
    directive. (If no shell can be found at all, DR-DOS IBMBIO.COM would
    display a prompt to enter the proper path to COMMAND.COM).
  The DR-OpenDOS 7.02+ COMMAND.COM is designed to also run on 3rd party
    operating systems like MS-DOS/PC DOS 3.31+, Windows 9x, and in DOS
```

```
    boxes of OS/2, Windows NT and 2000, and will for example also take
    advantage of long filenames.
SeeAlso: AX=5501h

Format of Microsoft COMMAND.COM file image signature:
Offset  Size  Description (Table 04100)
 00h  3 BYTEs E9h xxh xxh (JMP instruction)
 03h  BYTE  version
    bits 7-4: major version
    bits 3-0: minor version code
    ( 60h for MS-DOS 6.0 and PC-DOS 6.1;
      64h for MS-DOS 6.20, Japanese MS-DOS 6.2, Hangeul MS-DOS 6.2;
      66h for MS-DOS 6.22, PRC (Chinese) MS-DOS 6.22;
      70h for PC DOS 7, PC DOS 7 Y2K edition )
Notes:  Since DR DOS 6.0+, MS-DOS 7.0+, PTS-DOS, S/DOS, 4DOS/NDOS use an .EXE
    style shell file format, this signature is not met by any release of
    these shells. DR DOS 3.31-5.0 used a different jump (E8h) at offset
    00h.
  Older releases of MS-DOS/PC DOS all had a jump (E9h) at offset 00h, but
    other values at offset 03h. For reference here is a list of the
    values for some of these older shells:
      00h for PC DOS 1.10, Olivetti DOS 2.11, MS-DOS 5.0,
      Russian MS-DOS 5.0
      1Eh for MS-DOS 4.01
      BAh for PC DOS 3.10, 3.20, MS-DOS 3.30
--------l-2F5501-----------------------------
INT 2F U - DOS 5+ - ROM COMMAND.COM INTERFACE
  AX = 5501h
Return: ???
Note: used to determine whether the caller is the first instance of ROM
    COMMAND.COM
SeeAlso: AX=5500h
--------R-2F5600-----------------------------
INT 2F - INTERLNK - INSTALLATION CHECK
  AX = 5600h
  DX = magic value FFFFh
  BL = instance number (00h = any, 01h = first loaded, etc.)
  (BH = 00h and CX = 0000h, see note)
Return: AL = FFh if installed
      BL = instance number
      CX = version number (CL = major, CH = minor)
```

```
      DX = resident CS of driver, DX:0000h -> header (see #02845)
Notes:  INTERLNK was derived from Sewell Development Corporation's FASTLYNX.
  Some Microsoft software explicitly clears the BH and CX registers
    before calling this function.
BUG:  reportedly, the MS-DOS INTERSRV program (though not INTERLNK) causes
    serious FAT corruption and data loss when run under Novell DOS 7.
    INTERSRV apparently checks for DR-DOS by testing the "OS"
    environment variable for the value "DRDOS", which is no longer
    correct for PalmDOS, Novell DOS, or OpenDOS.  A workaround may be
    to explicitly set the OS variable to "DRDOS".
SeeAlso: AX=5601h,AX=5602h,INT 60/AX=0000h

Format of Interlnk device driver header:
Offset  Size  Description (Table 02845)
 00h  DWORD pointer to next driver, offset=FFFFh if last driver
 04h  WORD  device attributes (see #01647,#01648)
 06h  WORD  device strategy entry point
 08h  WORD  device interrupt entry point
 0Ah  8 BYTEs character device name "NUL2    "
 12h 165 BYTEs  ???
 B7h 67 BYTEs fully qualified Interlnk filename
 FAh  6 BYTEs ???
100h  DWORD pointer back to Interlnk filename at offset B7h
104h  8 BYTEs ???
10Ch  BYTE  total number of redirected drives
10Dh  BYTE  first local drive number (0=A:)
10Eh  BYTE  printer redirection (0=no, 1=yes)
10Fh  BYTE  ???
110h  3 BYTEs LPT1...3 status (0FFh=invalid)
113h 26 BYTEs remote drive number (0=A:, 0FEh=unused) (refer to note below)
12Dh 26 BYTEs always 0FEh ???
147h 26 BYTEs always 0FFh ???
Note: to obtain the remote drive number, subtract the value at offset 10Ch
    from the local drive number before indexing into the table at 113h
    (example: if local drives F, G, H are remote drives C, F, E then
    the first three bytes at offset 113h are 02h, 05h, 04h)
  for each instance of Interlnk, an extra device driver is loaded, but
    all have the same device name NUL2
SeeAlso: #01646 at INT 21/AH=52h
--------R-2F5601-----------------------------
INT 2F - INTERLNK - CHECK IF REDIRECTED DRIVE
```

```
  AX = 5601h
  DX = magic value FFFFh
  BH = drive number (0=A:)
  BL = instance number (00h=any, 01h=first loaded, etc.)
  (CX = 0000h, see note)
Return: (as for AL=00h if redirected drive)
Note: some Microsoft software explicitly clears the CX register before
    calling this function.
SeeAlso: AX=5600h,AX=5601h
--------R-2F5602-----------------------------
INT 2F - INTERLNK - CHECK IF PORT IN USE
  AX = 5602h
  DX = magic value FFFFh
  BL = instance number (00h=any, 01h=first loaded, etc.)
  CX = base port address of COM / LPT port to check
Return: (as for AL=00h if port in use for a redirected drive)
SeeAlso: AX=5600h
--------d-2F5700-----------------------------
INT 2F U - IOMEGA DRIVERS - INSTALLATION CHECK
  AX = 5700h
  BX = program ID??? (0201h used by GUEST.EXE)
  DX = 496Fh ('Io')
Return: AL = status
      00h not installed
      FFh installed
SeeAlso: AX=5701h,AX=5710h,AX=5711h,AX=5712h
--------d-2F5701-----------------------------
INT 2F U - IOMEGA DRIVERS - SECONDARY INSTALLATION CHECK
  AX = 5701h
  BX = program ID??? (0201h used by GUEST.EXE)
  DX = 496Fh ('Io')
Return: AX = 0001h if GUEST.EXE installed
SeeAlso: AX=5700h,AX=5710h,AX=5711h,AX=5712h
--------d-2F5710-----------------------------
INT 2F U - IOMEGA DRIVERS - GET DRIVER INFORMATION???
  AX = 5710h
  BX = program ID??? (0201h used by GUEST.EXE)
  DX = 496Fh ('Io')
Return: AX = ??? (BX ORed with ???)
  BX = ??? (internal variable)
  CX = ??? (internal variable)
```

```
  DX = ??? (CX ORed with ???)
SeeAlso: AX=5700h,AX=5701h,AX=5711h,AX=5712h
--------d-2F5711----------------------------
INT 2F U - IOMEGA DRIVERS - LOCK MEDIA IN DRIVE
  AX = 5711h
  BX = program ID??? (0201h used by GUEST.EXE)
  DX = 496Fh ('Io')
Return: CF clear if successful (storage medium in drive)
      AX = new lock count
  CF set on error (drive empty)
SeeAlso: AX=5700h,AX=5701h,AX=5710h,AX=5712h
--------d-2F5712----------------------------
INT 2F U - IOMEGA DRIVERS - UNLOCK MEDIA IN DRIVE / EJECT
  AX = 5712h
  BX = program ID??? (0201h used by GUEST.EXE)
  DX = 496Fh ('Io')
Return: AX = new lock count (00h = unlocked)
Note: if the lock count was already zero, the storage medium is ejected
    from the drive
SeeAlso: AX=5700h,AX=5701h,AX=5710h,AX=5711h
--------c-2F5758BX4858----------------------
INT 2F U - Helix Multimedia Cloaking - CACHECLK - INSTALLATION CHECK
  AX = 5758h
  BX = 4858h ('HX')
  DX = 4443h ('DC')
  CX <> 5758h
Return: BX = 6878h if installed
  DX = 6463h if installed
     CX = version (CH=major,CL=minor)
Program: CACHECLK is a 'Cloaked' disk cache by Helix Software
Note: returns with registers unchanged if CX=5758h on entry
SeeAlso: INT 16/AX=5758h/BX=4858h,INT 2F/AX=4310h"Cloaking"
--------X-2F5D00----------------------------
INT 2F U - PCMCIA - AWARD PCDISK - GET INFO FROM DRIVER ???
  AX = 5D00h
Return: ES:BX -> ???
Note: supported by Ventura Micro / Award PCDISK.EXE v1.02c PCMCIA/ATA driver
SeeAlso: AX=5D01h,INT 21/AX=440Dh"DOS 3.2+"
--------X-2F5D01----------------------------
INT 2F U - PCMCIA - AWARD PCDISK - PUT INFO INTO DRIVER ???
  AX = 5D01h
```

```
  ES:BX -> ???
Return: nothing
Note: supported by Ventura Micro / Award PCDISK.EXE v1.02c PCMCIA/ATA driver
SeeAlso: AX=5D00h,INT 21/AX=440Dh"DOS 3.2+"
--------s-2F60FFDL00-------------------------
INT 2F U - IPLAY v1.00b - INSTALLATION CHECK
  AX = 60FFh
  DL = 00h (function number)
  BX = 5344h ('SD')
  CX = 4D50h ('MP')
Return: AX = 4F4Bh ('OK') if installed
Program: IPLAY is the Inertia Player by Prime and Excalibur for .MODules
    (digitized music files)
Note: in version 1.00b, any value for DL except 01h invokes this function
SeeAlso: AX=60FFh/DL=01h
--------s-2F60FFDL01-------------------------
INT 2F U - IPLAY v1.00b - GET DATA SEGMENT
  AX = 60FFh
  DL = 01h (function number)
  BX = 5344h ('SD')
  CX = 4D50h ('MP')
Return: AX = data segment
Program: IPLAY is the Inertia Player by Prime and Excalibur for .MODules
    (digitized music files)
SeeAlso: AX=60FFh/DL=00h
--------v-2F6282-----------------------------
INT 2F U - PC Tools v7.0+ VDEFEND, VSAFE, VWATCH, DATAMON - SET ??? ADDRESS
  AX = 6282h
  CX:DX -> ??? or 0000h:0000h
  DI = segment of ??? record (see #02846) or 0000h/FFFFh to ignore
Return: BX = 0062h
Note: if CX:DX = 0000h:0000h on entry, the ??? address is not changed
    (DATAMON only)
SeeAlso: INT 13/AH=FAh"VSAFE",INT 21/AH=FAh"VDEFEND"

Format of VSAFE/VWATCH record:
Offset  Size  Description (Table 02846)
 00h  DWORD ???
 04h  WORD  offset of ??? in record's segment
    VSAFE 2.0 sets byte at +01h to 56h or 58h
    VWATCH 2.1 sets byte at +02h to 56h or 58h
```

```
 06h  2 BYTEs ???
 08h  BYTE  ??? (01h/other)
--------v-2F6284BX0000-----------------------
INT 2F U - PC Tools v7-8 DATAMON, v9+ DPROTECT  - INSTALLATION CHECK
  AX = 6284h
  BX = 0000h
  CX = 0000h
Return: AX = segment of resident code
  BX = 5555h
  CX = 5555h
Note: also supported by DOS 6 UNDELETE which is licensed from PC Tools
SeeAlso: AX=6284h/BX=0001h,INT 16/AX=FFA3h/BX=0000h
--------v-2F6284BX0001-----------------------
INT 2F U - PC Tools v7-8 DATAMON, v9+ DPROTECT - GET ???
  AX = 6284h
  BX = 0001h
  CX = 0001h
Return: AX:BX -> ??? data (see #02847)
  CX = BX
SeeAlso: AX=6284h/BX=0000h

Format of DPROTECT data for v9.0:
Offset  Size  Description (Table 02847)
 00h  5 BYTEs ???
 05h  WORD  resident code segment (may be segment of DWORD at +03h)
 07h  DWORD -> FAR function to sound alert tone
  ???
--------v-2F6284BX0002-----------------------
INT 2F U - PC Tools v7-8 DATAMON, v9+ DPROTECT - GET OPTIONS
  AX = 6284h
  BX = 0002h
  CX = 0002h
Return: AX = options (see #02848)
  BX = ??? (0000h for v9)
  CX = AX
  DX = BX
Note: also supported by DOS 6 UNDELETE which is licensed from PC Tools
SeeAlso: AX=6284h/BX=0000h,AX=6284h/BX=0003h

Bitfields for DATAMON/DPROTECT options:
Bit(s)  Description (Table 02848)
```

```
 1  ???
 12 disabled
 13 using Delete Sentry
 14 using Delete Tracker
--------v-2F6284BX0003-----------------------
INT 2F U - PC Tools v7-8 DATAMON, v9+ DPROTECT - SET OPTION??? FLAGS
  AX = 6284h
  BX = 0003h
  CX = flags (see #02849)
  DX = flags
      bit 15: ???
Note: v9 DPROTECT only checks bit 12 of CX, and ignores DX entirely
SeeAlso: AX=6284h/BX=0002h

Bitfields for DATAMON/DPROTECT CX flags:
Bit(s)  Description (Table 02849)
 3  ???
 5  ???
 10 ???
 12 disable DATAMON/DPROTECT
--------v-2F6284BX0004-----------------------
INT 2F U - PC Tools v8 DATAMON, v9+ DPROTECT - ???
  AX = 6284h
  BX = 0004h
  CX = 0004h
Return: AX = 5555h
  BX = ??? (0800h)
  CX = ??? (FCCCh for v8, FCCBh for v9)
--------V-2F6400-----------------------------
INT 2F - SCRNSAV2.COM - INSTALLATION CHECK
  AX = 6400h
Return: AL = installation state
      00h not installed
      FFh installed
Program: SCRNSAV2.COM is a screen saver for PS/2s with VGA by Alan Ballard
SeeAlso: INT 10/AX=5555h,INT 14/AX=AA01h
Index:  screen saver;SCRNSAV2
--------N-2F7000-----------------------------
INT 2F - License Service API - INSTALLATION CHECK
  AX = 7000h
  CX = license server index (0000h to 001Fh)
```

```
Return: AL = status
      00h not installed
      FFh installed
Notes:  The License Service API is being maintained by Microsoft but is being
    supported by a large number of companies including Apple, Banyan,
    DEC, HP, Lotus, Microsoft, Novell, Software Publishers Association,
    and Wordperfect (not a complete list!)
  Each license service provider must search for the next free index
    slot to use
SeeAlso: AX=7001h,AX=7003h,AX=7004h,AX=7005h
--------N-2F7001-----------------------------
INT 2F - License Service API - REQUEST LICENSE
  AX = 7001h
  CX = license server index (0000h to 001Fh)
  DS:DX -> SLSREQUEST structure (see #02850)
Return: AX = status
      0000h success
      else  provider error code
  ES:BX = provider specific handle for the license context
SeeAlso: AX=7002h,AX=7004h,AX=7005h

Format of License Service SLSREQUEST structure:
Offset  Size  Description (Table 02850)
 00h  DWORD (ret) status code
 04h  DWORD (ret) handle identifying context
 08h  DWORD (call) address of Publisher string
 0Ch  DWORD (call) address of Product string
 10h  DWORD (call) address of Version string
 14h  DWORD units required
 18h  DWORD address of comment string
 1Ch  DWORD address of SLSCHALLENGE structure (see #02851)

Format of License Service SLSCHALLENGE structure:
Offset  Size  Description (Table 02851)
 00h  DWORD algorithm (currently always 1)
 04h  DWORD secret to be challenged (1-255)
 08h  DWORD size of challenge in bytes (1-255)
 0Ch  N BYTEs challenge data
--------N-2F7002-----------------------------
INT 2F - License Service API - RELEASE LICENSE
  AX = 7002h
```

```
  CX = license server index (0000h to 001Fh)
  DS:DX -> SLSRELEASE structure (see #02852)
  ES:BX = provider specific handle for the license context
Return: AL = status
      00h not installed
      FFh installed
SeeAlso: AX=7001h,AX=7005h

Format of License Service SLSRELEASE structure:
Offset  Size  Description (Table 02852)
 00h  DWORD handle indentifying license context
 04h  DWORD total units consumed
 08h  DWORD address of comment string
--------N-2F7003-----------------------------
INT 2F - License Service API - UPDATE
  AX = 7003h
  CX = license server index (0000h to 001Fh)
  DS:DX -> SLSUPDATE structure (see #02853)
  ES:BX = provider specific handle for the license context
Return: AL = status
      00h not installed
      FFh installed
SeeAlso: AX=7004h,AX=7005h

Format of License Service SLSUPDATE structure:
Offset  Size  Description (Table 02853)
 00h  DWORD (ret) status code
 04h  DWORD (call) handle identifying license context
 08h  DWORD (call) total units consumed
 0Ch  DWORD additional units required
 10h  DWORD address of comment string
 14h  DWORD address of SLSCHALLENGE structure (see #02851)
--------N-2F7004-----------------------------
INT 2F - License Service API - GET ERROR
  AX = 7004h
  CX = license server index (0000h to 001Fh)
  DS:DX -> SLSGETERROR structure (see #02854)
  ES:BX = provider specific handle for the license context
Return: AL = status
      00h not installed
      FFh installed
```

```
SeeAlso: AX=7000h,AX=7001h

Format of License Service SLSGETERROR structure:
Offset  Size  Description (Table 02854)
 00h  DWORD (ret) status code
 04h  DWORD handle identifying license context
 08h  DWORD error code
 0Ch  DWORD buffer size in bytes
 10h  N BYTEs data buffer
--------N-2F7005-----------------------------
INT 2F - License Service API - QUERY LICENSE
  AX = 7005h
  CX = license server index (0000h to 001Fh)
  DS:DX -> SLSQUERY structure (see #02855)
  ES:BX = provider specific handle for the license context
Return: AL = status
      00h not installed
      FFh installed
SeeAlso: AX=7001h,AX=7002h

Format of License Service SLSQUERY structure:
Offset  Size  Description (Table 02855)
 00h  DWORD (ret) status code
 04h  DWORD handle identifying license context
 08h  DWORD information index
 0Ch  DWORD buffer size in bytes
 10h  N BYTEs data buffer
--------K-2F7041BX4B70-----------------------
INT 2F U - HP 200LX - KEY200 - INSTALLATION CHECK
  AX = 7041h
  BX = 4B70h
Return: BX = 7965h if keyboard remapper KEY200.COM installed
--------d-2F7200-----------------------------
INT 2F - SRDISK v1.30+ - INSTALLATION CHECK
  AX = 7200h
Return: AL = FFh if installed
      ES = segment of device driver header (see #02856)
Program: SRDISK is a freeware resizeable RAMdisk by Marko Kohtala
SeeAlso: AX=7201h

Format of SRDISK device driver header:
```

```
Offset  Size  Description (Table 02856)
 00h 10 BYTEs same as standard device driver header
      (see #01646 at INT 21/AH=52h)
 0Ah  BYTE  number of subunits (drives) supported by driver
 0Bh  3 BYTEs signature "SRD"
 0Eh  4 BYTEs memory type string ("XMS "/"EMS ")
 12h  4 BYTEs ASCII driver version string "N.NN"
 16h  BYTE  00h
 17h  BYTE  configuration format version (currently 00h or 01h)
 18h  WORD  offset of drive configuration data
SeeAlso: #01646
--------d-2F7201-----------------------------
INT 2F - SRDISK v2.02 - GET CODE/DATA SEGMENT
  AX = 7201h
Return: AL = FFh if installed
      ES = segment of device driver header (see #02856)
Program: SRDISK is a freeware resizeable RAMdisk by Marko Kohtala
SeeAlso: AX=7200h
--------!---Section--------------------------
```