DEVLOAD PROJECT.

CONTENTS.

SECTION.	TITLE.
1.	Project Writeup.
2.	DOS Function Reference.
3.	Structure of Internal DOS Tables.
4.	DEVLOAD Flowchart.
5.	DEVLOAD Source Code.
6.	NETLIST Source Code.
7.	REDIR Source Code.
8.	REDIRSHW Source Code.
9.	MYSHELL Source Code.
10.	WOMBATS Source Code.
11.	SHOWPARM Source Code.

A.E.B. 'A' Level Computing Examination (0643), Summer 1993.

Centre number: 18415. Candidate number: 3205.

PROJECT WRITEUP.

1. Definition of problem.

While I was a student at Norwich City College, I wished to use a copy of *Adobe Type Manager* on the College computers, as I was spending a lot of time preparing documents for the Diss Venture Scout Unit, including the South Norfolk District Scout Newsletter, which we had taken responsibility for. *ATM* is a piece of software, running under Windows, which allows the user to see text on screen exactly as it will look on the final printout, which makes desktop publishing very much easier. There was a copy of *ATM* somewhere on the College network, but I was not allowed access to it. I had to bring in my own copy from home, but for copyright reasons, I was not allowed to leave my copy on the College network drives. I had to find some suitable method of running the software without breaking the College network rules.

2. Analysis of problem.

The *Adobe Type Manager* software, along with the fonts I required, occupied a space of roughly 500 Kbytes. I needed this to be accessible at all times when it was in use, as Windows has no suitable facility for finding missing files, and tends to crash when it encounters such problems.

I could think of three approaches to the problem. All three involved an entire drive being dedicated to the software. They were:

2.1. Stacker drive.

One possible solution was to have as Stacker drive on the network. This would appear as a single compressed file unless mounted using the Stacker software, and would not be immediately recognisable as containing a piece of software.

This was not a suitable solution. Such a volume would have exceeded my data space limitation, and would not have gone unnoticed by the network managers. Also, the Stacker program comes in the form of a device driver, which must usually be loaded at boot time.

2.2. Floppy drive.

My first idea was to put the program onto a floppy disc. This worked, but it meant that I could not easily use the floppy drive for anything else. If I wanted to access any other floppy disc, I had to be very careful when I swapped discs, and it was safest to exit Windows to do so, which was time-consuming and irritating. Because I often worked on files at home, at work and at College, keeping them on a floppy disc all the time, I needed to access floppy discs a lot. This solution was not very suitable, and I only ran *ATM* when I really needed it, because of the problems it caused.

2.3. RAM drive.

This seemed like by far the best solution. The workstations I was using were all equipped with 4Mb RAM, so installing a 500Kb RAM drive would cause no memory problems. The software could be copied onto the RAM drive once when the machine was set up, and would remain in place until I logged off.

The main problem with this solution was again that the RAM drive software was in the form of a device driver which needed to be loaded at boot time. This is not easy to achieve on a workstation which does a remote boot from a server over the Ethernet network.

3. Possible solutions.

Having decided that a RAM drive was the best way of solving the problem, I was left with two possible ways of installing it, both of which involved quite a large amount of low-level work. These were:

3.1. Load RAM drive at boot time.

I asked a network manager about this, and he said it was not possible to do so. To achieve control of the machine at boot time, I needed to hack into the network, take copies of all the network drivers and put them all onto my own boot floppy. The difficult part was, of course, the first part.

3.2. Load RAM drive after boot time.

This solution would involve a lot of in-depth knowledge of the internal working of DOS and the structure of DOS tables. In the absence of any suitable technical reference manual, I would have to do all my own research, by taking apart DOS code. This involved even more work than the other possibility.

4. Solution number 1 - Load at boot time.

The network running on the workstations I was using was RM-NET 3.1. This is MS-NET compatible, which meant that my technical reference contained details of the DOS function calls necessary to connect and disconnect network drives.

The first step I took was to write, in C, a program to list the network devices connected to the system. I soon expanded it to include connection and disconnection of network devices, but it kept it's original name, `NETLIST.EXE`.

Now I had this method of listing connected devices and connecting more, I was more aware of what was happening within the system. I decided that it was time I found out the passwords for some of the network drives.

Working on the basis that they had to connected at some point in time by calling the DOS function to connect network devices, I set up a simple Trojan Horse to take over the DOS function interrupt (INT 21h). Whenever an INT 21h call was made with the `AX` register set to 5F02h (the function code for connecting a network device), the program recorded the drive connected, the shortname and the password in an internal buffer. The program was written in assembly language, and was called `REDIR.EXE`

I originally started by using `SYMDEB` to examine the program in memory and read the contents of the buffer, but I soon tired of that and wrote a program to display the copied information automatically. This was `REDIRSHW.EXE`, and I wrote it in C. For this I had to amend `REDIR.EXE` to include a signature, so that `REDIRSHW` could be sure the Trojan Horse was installed, and a pointer to the buffer, so that I wouldn't have to change `REDIRSHW.EXE` every time I changed the size of the Trojan Horse program.

The Trojan Horse worked perfectly, and soon I had found the shortnames and passwords of drives I hadn't previously known existed. Every server on the network had a drive named `PUBLIC`, and they all had the same password. This applied even to a server they hadn't told us about, meant for Computer Services use only. The `PUBLIC` drive on this contained copies of software not generally available on the network, for example Excel 4 and Superbase 2. I was encouraged by this discovery, and continued.

When logging off, the logon program usually just disconnects all network drives and asks for another user name and password for logon. If a TSR was installed, however, the machine was automatically rebooted, which meant that my Trojan Horse was lost every time I logged off from the system, making it difficult to gain access to many drives which I didn't usually have access to.

This problem was soon fixed, however. I used SYMDEB to examine the image of the logon program, XNETLIST, in memory, found all the checks performed and by-passed them. The program then didn't reboot when I logged off, but left my Trojan Horse running while the next user logged on. This was the point at which I began to enjoy myself.

When I examined the record of device connections after logging off and then back on again, I found that a connection had been made to a drive X:, and subsequently disconnected. I immediately reconnected to this drive and examined it. I found it to contain the version of DOS from which the machines booted, all the network device drivers, the logon program and also the user database containing all the passwords on the system. This was exactly what I had been looking for.

I examined the startup procedure carefully, to avoid making any mistakes and causing problems when I made my boot floppy.

The stations boot from the closest server on the network. and immediately connect drive X: to the network boot drive. A copy of CONFIG.SYS is then executed, depending on the type of the workstation. This loads all the device drivers necessary for normal operation of DOS and the network. NET.EXE is then called as the DOS shell. This connects a network session and then passes control to XNETLIST.COM, which is basically a loop which does the following:

1. Connect drive X:
2. Run XNETLIST.EXE *(see below.)*
3. Disconnect all drives, rebooting if can't
4. Reboot if INT 21h vector changed.
5. Reboot if amount of free memory has changed.
6. Various other checks, rebooting if changes.
7. Loop to part 1.

XNETLIST.EXE is the program which handles to logon procedure, by asking for a user name and password and looking them up in the user database. If they are correct, the necessary network drives are connected and a copy of COMMAND.COM is invoked.

Once I was sure of the boot procedure and was confident that I could make a network boot floppy without making any mistakes that might disrupt the network, I copied all the network drivers to a DOS 5 boot disc along with the CONFIG.SYS file for the workstations I was using, and attempted to boot from my own floppy, This worked fine with an exact copy of the CONFIG.SYS, so I added a RAM drive, and was delighted to find that I had solved the problem.

I soon cut out the network security programs from the boot procedure, after first examining them to ensure there would be no adverse effects caused by their removal.

My next step was to by-pass the normal logon procedure. I got bored of typing my name and password, so I wrote MYSHELL.EXE in C to log on automatically. This connected me to both my own user space and the Student Union user space, in which I occasionally had to work

The only problem with this was that many of the shortnames and passwords were eight-digit random number which were changed every week. About once a week, I had to log on normally and catch all the shortnames and passwords again. This only took a couple of minutes, though, and so I didn't go back to the normal logon program.

A more important problem with my solution was that all the workstations were set up to remote boot from the network, and only attempt to boot from a floppy disc if the network was not available. This meant that I had to unplug my machine from the network every time I booted it. I had, however, seen that the network managers possessed boot discs which the machines booted from in

preference to the network. The discs obviously contained a signature, probably in the boot sector, marking them as having priority over a network boot. The machines always checked the floppy drive for this signature before booting from the network.

In order to find out exactly what this signature was, I took home a complete copy of the machine's ROM BIOS and started to go through the INT 19h procedure, which is the ROM bootstrap.

Before I completed this, however, I encountered another obstacle. I was asked by the network managers to refrain from using my own boot floppy.

I explained exactly what I was using it for, and asked the Development Team Manager about the possibility of loading device drivers from the command line. He replied that he'd tried it once, but had given up on it.

Naturally, I took this as a challenge, and threw myself into solving the my second possible solution; loading device drivers from the command line.

5. Solution number 2 - Load from command line.

This involved writing a program to install a RAM drive into the DOS internal tables after boot time. I decided to take it a step further and make it install all device drivers.

6. Program requirements.

Before I could write a program to install device drivers, I first had to work out how it was done. My main problem was that I had no knowledge of the internal DOS tables into which I had to enter all the drive statistics. I was aware that the procedure for loading a device driver went as follows:

1. Load driver file into memory.
2. Pass the driver an `INITIALISE` command.
3. Examine the amount of memory the driver requires to stay in memory.
4. If this is zero, exit. The driver has completed it's task and does not need to stay resident.
5. Allocate the correct amount of memory to the driver.
6. Link the driver into the device driver chain.
7. If it is a block device, install all the blocks into the relevant internal tables.

Steps 1-5 are simple operations, the information and procedures necessary being available in any semi-decent technical reference for DOS.

Step 6 involves the use of an undocumented DOS function, function 52h. This gets a pointer to a DOS table 'invar'. Each device driver started with a DWord pointer to the next device driver in the chain. The first device in the chain is the NUL device, which is located at invar + 0022h. To link the new driver into the chain, the NUL device must be changed to point to the new device, and the new device must be made to point to where NUL used to point to. This installs the driver in the same place in the chain as if it had been installed in `CONFIG.SYS`.

At this point, I had no idea about how to implement step 7, so I concentrated on getting the rest to work. This was enough to install character devices.

I started by doing it all by hand in `SYMDEB`, and once I was sure of the procedure, I wrote the first version of `DEVLOAD`. As can be seen from the alteration list, it was very primitive. I soon made many changes to improve the functionality and user-friendliness of the program, all of which are listed in the source code.

At this point, I wrote `SHOWPARM.SYS` to display exactly the command line passed to the driver by DOS. This showed that when no parameters are given to a device driver, DOS inserts a space after

the filename. DOS also converts the whole of the command line to upper case. For compatibility, I made DEVLOAD do the same. in version 2.1.

Once I had cleaned up the program to a reasonable extent, I decided that it was time to include block devices.

At this point , I was stuck. I had no idea of how to continue, so I was forced resort to the age-old principle 'If you don't know what you're doing, copy someone else.' I realised that there was already a section of code in the system that installed blocks into the internal tables. It is part of the SYSINIT module, the section of code which sets up DOS immediately after booting and is responsible, among other things, for the parsing of CONFIG.SYS and the loading of device drivers.

I wrote a device driver, calling it WOMBATS.SYS in the absence of any better ideas, which grabbed the entire SYSINIT module from it's location in high memory into a buffer and reported the return address (the address which the driver was invoked from.) This allowed me to examine the SYSINIT code at my own leisure. After a couple of days inspecting the SYSINIT code, I managed to complete my list of what step 7 in my original flowchart actually entailed:

7.1. Check sector size against maximum in system.
7.2. It too large, don't install driver.
7.3. Check LASTDRIVE array. Don't install if the array is not large enough to fit another drive.
7.4. Insert validity flag and pointer to block header into LASTDRIVE array.
7.5. Create new block header for drive.

The first four steps were easy enough to implement, but step five was partially done by another undocumented DOS function, number 53h. This is used to expand the BPB returned by the device driver into the block header. Step 7.5 was as follows:

7.5.1. Allocate enough memory for the new block header.
7.5.2 Insert the absolute block number, the block number in the device, and the pointer to the device driver.
7.5.3. Link the new block header into the chain by making the one that was previously at the end of the chain point to it and setting the offset of the pointer in the new one to 0FFFFh, signalling the end of the chain.
7.5.4. Use function 53h to insert the rest of the data into the block header.

I now had a complete flowchart, at least at a greatly simplified level, for the installation of device drivers. I proceeded to update DEVLOAD to version 2.0, including block device installation. Again, it was primitive to start with, but I soon updated the program as and when I realised what could be done to improve it.

Because the program started off as a test for an idea and developed from there, there was no flowchart and program plan, but it evolved as it went. When I started to write it, I had no idea of the problems I would encounter, so attempting to write a program plan and flowchart would have been totally pointless.

As it was, the program stayed reasonably well-structured, in spite of the continual alterations. Even so, I decided to write out a flowchart, including a few new ideas, and do a complete rewrite, putting it in .EXE format and calling it version 3.0.

This was fine in theory, but almost as soon as I had finished the rewrite, I finally managed to work out how to relocate the PSP and save the 60h bytes of memory below the driver that I had always before had to leave for the PSP. Having done this at last, after about a year, on and off, of

experimenting, I made many changes to the memory allocation procedure, as can be seen in the alteration list for version 3.0.

Having never found a device driver file containing two drivers with blocks to install, I had a few bugs in the install procedure for block devices which had never shown up. I put together two copies of DRIVER.SYS into a single file, and this highlighted a few problems which I also fixed..

The only other change which I made after writing version 3.0 was to move the main program entry point to a location above LASTBYTE. This results in absolutely no change to the program code, it just means that the code which has already been executed and is no longer required is not relocated to the top of memory.

The final flowchart for DEVLOAD.EXE is included after the appendices.

7. Testing.

The program has largely been tested as it was developed, with bugs being fixed as they came up. Usually, each time I got the program working properly or added a new function, I discovered more bugs which needed to be fixed, or at least I had new ideas about what I could make it do next.

The following is a demonstration of DEVLOAD being used to load an expanded memory driver. The memory map function, MEM.EXE, is called both before and after loading EMM.SYS to confirm the addition of the expanded memory function. DEVICES.COM is called last to show the position of EMM.SYS in the device chain.

```
C:\ >mem


    655360 bytes total conventional memory
    655360 bytes available to IBM DOS
    495072 largest executable program size

C:\ >devload /v emm.sys
DEVLOAD.EXE v3.0  (C) 1992, 1993 David Woodhouse.
 Loads device drivers from the command line.

Filename     : C:\MSDOS\EMM.SYS
Load address : 2718:0000


Expanded Memory Manager  Version 4.0S
Size =  384 KB, Page Frame = D000
Port(s) = 0208 0258

Init function return status : 0100
1 character device installed.
Size of driver (paragraphs) : 021A
Interrupt vectors changed   : 67h.
Driver staying resident.

C:\ >mem


    655360 bytes total conventional memory
    655360 bytes available to IBM DOS
    486448 largest executable program size
```

```
    393216 bytes total EMS memory
    393216 bytes free EMS memory

C:\ >devices

DEVICES.COM v2.2 (C) 1991, 1992 David Woodhouse.
 Lists drivers in DOS device chain.

Table invar located at 011C:0026

Device    Attr. Str.  Int.   Address      .SYS file
NUL       8004  0DC6  0DCC  011C:0048
EMMXXXX0  8000  0016  0023  2718:0000  EMM
Block:03  0002  0039  004F  1C10:0000  NETUNITS
Block:02  4842  0126  0131  0B61:0000  STACKER
CON       8013  06F5  0700  0070:0023
AUX       8000  06F5  0721  0070:0035
PRN       A0C0  06F5  0705  0070:0047
CLOCK$    8008  06F5  0739  0070:0059
Block:04  08C2  06F5  073E  0070:006B
COM1      8000  06F5  0721  0070:007B
LPT1      A0C0  06F5  070C  0070:008D
LPT2      A0C0  06F5  0713  0070:009F
LPT3      A0C0  06F5  071A  0070:00B8
COM2      8000  06F5  0727  0070:00CA
COM3      8000  06F5  072D  0070:00DC
COM4      8000  06F5  0733  0070:00EE

C:\ >
```

I have no character device drivers that could easily be demonstrated. I have tested `ANSI.SYS` and found it to work when loaded by `DEVLOAD`, but cannot easily show screen dumps of this. The next example is of `DEVLOAD` being used to load the SuperStor compression software which I use to compress my hard drive. The `VOL` command is used to show the absence and later the presence of the specified drive. Whenever I install either SuperStor or Stacker on any system, I always keep a copy of `DEVLOAD` in the uncompressed part of the drive, for loading the compression device driver after booting from a floppy disc.

```
C:\ >vol d:

Invalid drive specification

C:\ >devload /v sstordrv.sys

DEVLOAD.EXE v3.0  (C) 1992, 1993 David Woodhouse.
 Loads device drivers from the command line.

Filename     : C:\SSTORDRV.SYS
Load address : 171B:0000

SuperStor Data Compression Driver (DRI) 1.06
Copyright (C) AddStor Inc. 1991. All rights reserved.

Block header for drive D: at 21EA:0000
Init function return status : 0100

Last drive in use : D:
```

```
Last drive avail. : Z:
1 block installed.
Size of driver (paragraphs) : 0AD2
Interrupt vectors changed   : 21h, 26h.
Driver staying resident.

C:\ >vol d:


 Volume in drive D is SuperStor

C:\ >
```

The last example presented here is DEVLOAD attempting to load a block device driver when the LASTDRIVE array is already full. This demonstrates how it asks whether to terminate installation.

```
C:\ >devload /v driver.sys /d:2

DEVLOAD.EXE v3.0  (C) 1992, 1993 David Woodhouse.
 Loads device drivers from the command line.

Filename     : C:\MSDOS\DRIVER.SYS
Load address : 2E32:0000

Loaded External Disk Driver for Drive K

Init function return status : 0103
1 block(s) not installed - LASTDRIVE= parameter in CONFIG.SYS too small.

No blocks or INTs installed - terminate (Y/N) ? y
Size of driver (paragraphs) : 0000

C:\ >
```

8. Documentation.

User documentation is included within the program. The usage is very simple, and the standard help command ('DEVLOAD /?') will display the available options. Even if DEVLOAD is invoked with no arguments, it will tell the user how to get help.

There is little need for extensive system documentation in this program.. Any person editing this program must, because of it's nature, have an extensive knowledge of 8086 assembly language and the DOS operating system. To anyone with this capability, much of the program will be self-explanatory, and only rudimentary comments in the source code are necessary. However, I have included more documentation than I would usually provide, both because this is performing complex tasks in machine code and it is easy to lose track of register usage, and because I know it is intended to be examined by other people than just myself.

9. Critical appraisal.

I have had many criticisms of the operation of the program, but over the past year I have fixed just about all of them. There are still changes I would like to make, and probably will in the near future when I have completed enough research to make them possible. I have worked out how to

implement support for SETVER.EXE, and am waiting for MS-DOS 6 to arrive so that I can check it's compatibility before another update. I have also worked out how to change the size of the LASTDRIVE array in DOS 4 and higher. This I will incorporate soon as well.

Loading device drivers into UMBs will involve a lot more experimenting and probably examination of SYSINIT, so will probably not happen in the near future. I'll get round to it, though, and by then I'll have come up with something else to add or change. I might end up trying to get it to load devices once Windows is running, and make them available to all programs under Windows. That'd keep me amused for a while.

APPENDIX 1 - DOS FUNCTION REFERENCE.

The DOS function call is INT 21h. The function number is passed in the AH register. This appendix lists all the DOS functions used in any of the programs included with this project.

Function 02h - Character output.

IN:	DL	ASCII character to print.
OUT:	Nothing.	

Prints the character in DL to the standard output device (usually CON: if not redirected.)

Function 08h - Character input without echo.

IN:	Nothing.	
OUT:	AL	Character from device.

Gets the next character from the standard input device (usually CON: if not redirected.)

Function 09h - Output character string.

IN:	DS:DX	--> ASCII$ string to print.
OUT:	Nothing.	

Prints the dollar-terminated string to the standard output device.

Function 25h - Set INT vector.

IN:	AL	INT number.
	DS:DX	INT vector.
OUT:	Nothing.	

Sets the specified interrupt vector to the value held in DS:DX.

Function 30h - Get version number.

IN:	Nothing.	
OUT:	AL	Major version number.
	AH	Minor version number.

Used to return the current DOS version number.
NOTE: DOS version 1 returns zero in AL.

Function 31h - Terminate and stay resident (TSR).

IN:	AL	Program return code.
	DX	Memory to reserve in paragraphs.
OUT:	Nothing.	

Terminate program but remain in memory using DX paragraphs.

Function 35h - Get INT vector.

IN:	AL	INT number.
OUT:	ES:BX	INT vector.

Sets ES:BX to the value of the specified interrupt vector.

Function 37h - Get / set switch character (undocumented.)

IN:	AL	00h	Get switch character.
		01h	Set switch character.
	DL	Switch character if AL=01h	
OUT:	DL	Switch character if AL=00h	

Gets or sets the current character to be used as a switch on the command line (usually '/'.)

Function 43h - Get / set file attributes.

IN:	AL	00h	Get file attributes.
		01h	Set file attributes.
	CX	New attribute if AL=01.	
	DS:DX	--> ASCIIZ filename.	
OUT:	CX	File attribute if getting attribute.	
	AX	Error code if error.	
	CARRY	Set if error.	

Gets or sets attributes for the file whose name is pointed to by DS:DX. Can be used to check for file's existence.

Function 48h - Allocate memory.

IN:	BX	Number of paragraphs required.
OUT:	AX	Segment of block if successful.
		Error code if error.
	BX	Size of largest available block if error.
	CARRY	Set if error.

Attempts to allocate memory to the current procedure.

Function 49h - Release memory.

IN:	ES	Segment of block to be released.
OUT:	AX	Error code if error.
	CARRY	Set if error.

Attempts to release the memory block pointed to by ES back to the main pool.

Function 4Ah - Modify memory allocation.

IN:	ES	Segment of block to be modified.
	BX	New size in paragraphs.
OUT:	AX	Error code if error.
	BX	Size of largest available block if error.
	CARRY	Set if error.

Attempts to change the size of the specified memory block to the length in BX.

Function 4Bh - Execute program.

IN:	AL	00h	Load and execute program.
		03h	Load overlay.
	DS:DX	--> ASCIIZ filename.	
	ES:BX	--> Parameter block.	

For AL=03h, parameter block consists of:

+00	Word	Segment address to load overlay.
+02	Word	Relocation factor.

Loads program into memory and then executes if AL=00h.

Function 4Ch - Terminate with return code.

IN:	AL	Program return code.
OUT:	Nothing.	

Terminate program and return code in AL to parent.

Function 50h - Set current PSP (undocumented.)

IN:	BX	Segment address of new PSP.

Sets the current PSP to the one specified in BX.

Function 52h - Get ‘invar’ pointer (undocumented.)

IN:	Nothing.	
OUT:	ES:BX	--> Invar.

Gets pointer to DOS table ‘invar’. See Appendix 2 for details.

Function 53h - Expand BPB to block header (undocumented.)

IN:	DS:SI	--> BPB to expand.
	ES:BP	--> Block header to fill in.
OUT:	Nothing.	

Fills in block header with information from BPB.

Function 55h - Create child PSP (undocumented.)

IN:	DX	Segment address of new PSP.
	SI	Value to put in new PSP:0002 (top of mem segment.)
OUT:	Nothing.	

Creates a child PSP at the address specified.

Function 58h - Get / set allocation strategy.

IN:	AL	00h	Get strategy.
		01h	Set strategy.
	BX	New strategy code if AL=01h.	
OUT:	AX	Strategy code if AL was 00h	

Strategy codes:

0000	- First fit.
0001	- Best fit.
0002	- Last fit.

Gets or sets memory allocation strategy for subsequent memory requests.

Function 5Eh, subfunction 00h - Get machine name.

IN:	AL	00h	
	DS:DX	--> Buffer to receive string.	
OUT:	AX	Error code if error.	
	CH	00h	if name not defined.
		>00h	if name defined.
	CL	NETBIOS name number if CH>0	
	CARRY	Set if error.	

Gets machine name (MS-NET only.)

Function 5Fh, subfunction 02h - Get redirection list entry.

IN:	AL	02h
	BX	Redirection list index.
	DS:SI	--> Buffer to receive 16-byte device name.
	ES:DI	--> Buffer to receive 128-byte shortname.
OUT:	CARRY	Set if error.
	AX	Error code if error.

If not error:

	BH	Device status flag.
		bit 0 0 device valid.
		1 device invalid.
	BL	Device type.
		03h Printer.
		04h Drive.
	CX	Stored parameter value.
	DX, BP	Destroyed.

Gets entry number BX in the list of redirections.

Function 5Fh, subfunction 03h - Redirect device.

IN:	AL	03h
	BL	Device type.
		03h Printer.
		04h Drive.
	CX	Parameter to save for caller.
	DS:SI	--> ASCIIZ local device name.
	ES:DI	--> ASCIIZ shortname followed by ASCIIZ password.
OUT:	CARRY	Set if error.
	AX	Error code if error.

Attempt to connect device to network shortname, using given password.

Function 5Fh, subfunction 04h - Cancel redirection.

IN:	AL	04h
	DS:SI	--> ASCIIZ local device name.
OUT:	CARRY	Set if error.
	AX	Error code if error.

Disconnect device from network.

Function 60h - Expand filename.

IN:	DS:SI	--> Source pathname.
	ES:DI	--> Buffer to hold destination pathname.
OUT:	CARRY	Set if error.
	AX	Error code if error.

Gives true pathname, taking into account current drive and directory. Gives error if goes above root (i.e. too many '\..\').

Function 62h - Get current PSP.

IN:	Nothing.	
OUT:	BX	Segment address of current PSP.

Gets the segment address of the PSP of the current process.

APPENDIX 2 - STRUCTURE OF INTERNAL DOS TABLES.

Invar.

Offset	Size	Item
-02h	Word	Segment of first memory arena header.
00h	DWord	--> First block header.
04h	DWord	--> FILES array.
08h	DWord	--> CLOCK$ driver.
0Ch	DWord	--> CON driver.
10h	Word	Max. bytes per sector.
12h	DWord	--> First disk buffer.
16h	DWord	--> LASTDRIVE array.
1Ah	DWord	--> FCB table.
1Eh	Word	Size of FCB table.
20h	Byte	No. of block devices.
21h	Byte	LASTDRIVE value.
22h		NUL device starts here.

Device Driver.

Offset	Size	Item
00h	DWord	--> Next driver in chain (x:FFFF means end.)
04h	Word	Device attributes.
06h	Word	Device strategy routine offset.
08h	Word	Device interrupt routine offset.
0Ah	8 Bytes	Device name padded with spaces.

Bios Parameter Block (BPB.)

Offset	Size	Item
00h	Word	Bytes per sector.
02h	Byte	Sectors per cluster.
03h	Word	Reserved sectors.
05h	Byte	Number of FATs.
06h	Word	Max. root directory entries.
08h	Word	Total number of sectors.
0Ah	Byte	Media descriptor byte.
0Bh	Word	Sectors per FAT.

Block Header.

Offset	Size	Item
00h	Byte	Absolute block number.
01h	Byte	Block number in device.
02h	Word	Bytes per sector.
04h	Byte	Sectors per cluster - 1.
05h	Byte	Cluster to sector shift (how far to shift left bytes/sector to get bytes/cluster.)
06h	Word	Number of reserved sectors.
08h	Byte	Number of FATs.
09h	Word	Number of root directory entries.
0Bh	Word	Sector no. of first data.
0Dh	Word	No. of clusters + 1.
0Fh	Word	Sectors per FAT. *NOTE: This is a single byte in DOS 3, so all offsets from here onwards must have one subtracted from them.*
11h	Word	First sector of root directory.
13h	DWord	--> Device driver for this block.
17h	Byte	Media descriptor byte.
18h	Byte	0FFh means must rebuild.
19h	DWord	--> Next block header in chain (x:FFFF means end.)
1Dh	Word	?
1Fh	Byte	?
20h	Byte	?

Arena Header.

Offset	Size	Item
00h	Byte	'Z' if last block, else 'M'.
01h	Word	Segment owner.
03h	Word	Segment size.
05h	3 Bytes	Unused.
08h	8 Bytes	Owner name (if self owned.) Terminated with NULL byte if less than eight characters.

Program Segment Prefix (PSP.)

Offset	Size	Item
00h	Word	Program exit point (INT 20h.)
02h	Word	Memory size in paragraphs.
04h	Byte	Unused.
05h	5 Bytes	Far call to DOS function handler.
0Ah	DWord	Old INT 22h vector.
0Eh	DWord	Old INT 23h vector.
12h	DWord	Old INT 24h vector.
16h	Word	Parent PSP segment.
18h	14h Bytes	Open files (0FFh = unused.)
2Ch	Word	Environment segment.
2Eh	DWord	Far ptr to SS:SP.
32h	Word	Max. open files.
34h	DWord	--> Open files table (usually PSP:0018h.)
38h	8 Bytes	?
40h	Word	Version number reported to this process (DOS 5+.)
42h	0Eh Bytes	?
50h	3 Bytes	DOS function dispatcher (INT 21h, RETF.)
53h	Word	Unused.
55h		FCB #1 extension.
5Ch		FCB #1.
6Ch		FCB #2.
80h	80h Bytes	Command line tail.

APPENDIX 3 - DEVLOAD FLOWCHART.

1. Initialisation.

1.1. Set up segment registers.
1.2. Get PSP segment.
1.3 . Print initial message.
1.4. Check DOS version.

2. Check command line.

2.1 If no parameters given, print error and exit.
2.2. Deal with switches.
2.3. Search for file using PATH if no path specified.
2.4. If file not found, print error and exit.
2.5. Expand filename to full pathname using func. 60h.

3. Relocate.

3.1. Get allocation strategy.
3.2. Reduce main allocation to minimum.
3.3. Set allocation strategy to highest fit.
3.4. Request chunk at top of memory for PSP, program and stack.
3.5. Reset allocation strategy to old value.
3.6. Move PSP to top of memory.
3.7. Move program to top of memory.
3.8. Give ownership of top of memory segment to itself.
3.9. Change stack to top of memory.
3.10. Make top PSP current.
3.11. Transfer execution to top of memory.

4. Allocate lower segment of memory.

4.1. Change stored PSPSeg.
4.2. Release old PSPSeg.
4.3. Release old environment.
4.4. Grab all free memory.
4.5. Store DvcSeg.

5. Load driver.

5.1. Disable break.
5.2. Copy all INT vectors.
5.3. Parse command line.
5.4. Print filename.
5.5. Load driver using func. 4Bh
5.6. Print load address.

5.7. Get invar pointer.
5.8. Get max sector size.
5.9. Get LASTDRIVE, LastDrUsed.

6. Execute driver.

6.1. Set DS:SI --> DvcSeg:0000.
6.2. Set ES:BX --> device after NUL.
6.3. InstallDevice until no more left.
6.4. Print LASTDRIVE error if necessary.

7. Clear up.

7.1 . Calculate size of driver to keep.
7.2. Offer abort if nothing installed.
7.3. Print LASTDRIVE and LastDrUsed.
7.4. Print number of devices installed.
7.5. Insert new LastDrUsed into invar.
7.6. Print driver keep size.
7.7. If keep size is zero, exit.
7.8. Allocate driver memory required.
7.9. Print INT vectors changed.
7.10. Link from NUL device.
7.11. Put ownership and device name into driver's arena header.
7.12. Exit program.

INSTALLDEVICE ROUTINE.

1. Store driver addresses.

2. Print CrLf.

3. Make request header.

- **3.1. Insert next block number.**
- **3.2. Insert 'INIT' command.**
- **3.3. Insert default break address = DvcSeg:0000.**
- **3.4. Insert default no blocks in device.**
- **3.5. Insert ptr to command line tail.**

4. Call device routines.

5. If character device, increase count of said.

6. Check driver length.

- **6.1. Get driver length.**
- **6.2. If grown and blocks already installed, error.**
- **6.3. Else if grown and blocks not installed, store new value.**

7. Install blocks.

- **7.1. Check number of units in driver.**
- **7.2. If none, goto 8.**
- **7.3. Zero block number count in this device.**
- **7.4. ES:BP --> new block header.**
- **7.5. DS:BX --> BPB pointer array.**
- **7.6. DS:SI --> next BPB from pointer array.**
- **7.7. Check sector size.**
- **7.8. Check LASTDRIVE.**
- **7.9. Store abs. block number, block number in device.**
- **7.10. Make last block header in chain point to the new one.**
- **7.11. Change pointer to ChainEnd.**
- **7.12. Print new block header address.**
- **7.13. Fill in LASTDRIVE array.**
- **7.14. Finish filling in block header.**
- **7.15. Expand BPB to block header using func. 53h.**
- **7.16. Increase ES:BP by size of block header.**
- **7.17. Increase BlocksDone.**
- **7.18. If blocks left in this device, loop to 7.6.**

7.19. Print INIT return status.

8. Link driver.

8.1. Push address of next in file.
8.2. Point new driver to old driver.
8.3. Pop address of next in file.
8.4. Convert segment if necessary.
8.5. Set ZERO flag on whether last in file.
8.6. RETURN.

APPENDIX 4 - DEVLOAD SOURCE CODE.

```
;.....................PROGRAM HEADER FOR PROJECT BUILDER...................


;       TITLE   DEVLOAD         to load device drivers from command line.
;       FORMAT  EXE
;       VERSION 3.0
;       CODE    80x86
;       OPTIONS /ML
;       TIME    CHECK
;       DATE    12/3/92 - 21/4/93
;       AUTHOR  David Woodhouse
;        (C) 1992, 1993 David Woodhouse.

;EXPLANATION...

;       The program first relocates itself to the top of the available memory
;       and then loads the driver below itself. If the driver loads happily, it
;       is then executed. If it asks for memory to be reserved for it, it is
;       linked into the device chains and the program TSRs keeping the required
;       amount of memory, otherwise a normal exit occurs.


;............................ALTERATION LIST.............................


;Version 1.0            12/3/92
;       Basics, not user-friendly, only supports character devices so far.
;                       13/2/92
;       Change to using EXEC (4Bh) function to load - now loads .EXE files
;       Take length to TSR from device return, not file length.
;       Don't TSR if not required.


      ;Version 1.1           17/3/92
;       Complete rewrite of initialisation routine.
;       Allows for more than one device per file.
;       Help message added.
;       EXEC failure now explains error codes, rather than just giving no.
;                       19/3/92
;       Print error message and exit if version < 4.
;       Loads device at PSP+6, not PSP+10h (overlay FCBs and command tail).
;       Stack moved down by 8 paras, not truncated to 80h bytes.
;                       20/3/92
;       Release environment block before TSR.


;Version 2.0            21/3/92
;       Use INT 21h, function 53h. Can now load block devices.
;       Use segment in break address, don't assume same as driver segment.
;       Ask whether to terminate if can't install any blocks.
;       Disable ctrl-break.
;                       22/3/92
;       Check sector size before installing block devices.
;       Disable break with INT 1Bh, as well as INT 23h - stops ^C appearing.
;               (taken out in v2.1) - causes problems if driver changes it.
;       Don't use INT 10h - all output via INT 21h - can be redirected.
;       Print drive letter with block header address.
;       Print LastDrive message after installing block headers, not before.
;       Change program name in arena header to device filename (for MEM.EXE).
;       Support for DOS 3 added (now works with at least DOS 3.1 onwards).
;       Print driver's load address.


;Version 2.1            24/3/92
;       Bug fix - drivers requesting memory offset FFF1 - FFFF now works OK.
;                       25/3/92
```

```
;       Now uses func 60h to expand filename before printing it.
;       Bug fix - changing INT 1B lost vector if driver altered it, so don't.
;       Display INT vectors changed by driver.
;                       26/3/92
;       Converts params to upper case before passing to driver, like DOS does.
;       Also adds space after filename if no parameters given.
;       Change data at end to ?? rather than 00 - smaller .COM file.


;Version 2.2            26/10/92
;       When no blocks installed, checks whether INT vectors changed before
;       asking whether to terminate. Still not foolproof, but better.
;       Cosmetic fix - `$' now comes after CR/LF on `None.' for INT vectors.
;       Bug fix - Check all INT vectors (200h words, not 200 words!)


;Version 3.0            11/4/93 - 20/4/93
;       Complete rewrite from scratch.
;       Only relocate if going to try EXEC - saves losing F3.
;       Use path to find driver if not in specified directory.
;       Convert to .EXE program.
;       Relocate PSP to top of memory as well as code and stack.
;       Release environment before requesting memory for driver.
;       Don't even attempt to load driver if not enough memory available.
;       Use highest memory required return.
;       Link drivers in correct order.
;       Add /Q (quiet mode) option.
;       Disperse comments ad nauseum.
;                       21/4/93
;       Add /V (verbose mode) option.
;       Move lastdrive report to end.
;       Move abort request to end.
;       Add count of character devices installed.
;       Put entry point after LASTBYTE - smaller relocated code.


;.............................IMPROVEMENT IDEAS............................


;       Load into Upper Memory Blocks (like 'DEVICEHIGH='.)
;       Work from batch files like CONFIG.SYS.
;       Change size of LASTDRIVE array by reallocating.
;       Add support for SETVER.EXE.


;................................DEFINES.....................................

STACKLEN        equ     200h
SMALLESTDRIVER  equ     100h
QuietFlag       equ     80h
VerboseFlag     equ     40h
AutoFlag        equ     20h

;............................CODE (at last)..............................

CSeg    segment public  byte    'CODE'

        org     0

        assume  cs:CSeg, ds:CSeg, es:CSeg

;.............CODE TO BE EXECUTED ONCE RELOCATED TO TOP OF MEMORY............

        ;DS:TopCSeg, ES:TopPSPSeg

        ;Get old PSP segment, store new PSP segment.

relocated:      push    es
                mov     es,PSPSeg
                pop     PSPSeg
```

```
        ;Push segment of environment.

        ;DS:TopCSeg, ES:PSPSeg

                push    es:[002Ch]

        ;Release old PSP segment.

                mov     ah,49h
                int     21h
                jnc     relPSPok

        ;Failed to release PSP - print error.

                mov     dx,offset RelPSPErrMsg
                call    PrintError
                mov     ah,9
                int     21h
                mov     dx,offset CrLfMsg
                mov     ah,9
                int     21h

        ;Release environment segment.

        ;DS:TopCSeg, ES:CSeg

relPSPok:       pop     es
                mov     ah,49h
                int     21h
                jnc     relenvok

        ;Failed to release environment - print error.

                mov     dx,offset RelEnvErrMsg
                call    PrintError
                mov     ah,9
                int     21h
                mov     dx,offset CrLfMsg
                mov     ah,9
                int     21h

relenvok:       push    cs
                pop     es

        ;Find out how much memory is available by asking for stupid amounts.

                mov     ah,48h
                mov     bx,0FFFFh
                int     21h

        ;In MS-DOS version 6 and below, this will always fail, but check
        ;the return status just in case it does give what we asked for.

                jnc     grablowok

        ;BX now holds the maximum amount of memory available.
        ;Don't install if less than 4K bytes available.

                cmp     bx,SMALLESTDRIVER
                ja      sizeok
                mov     bx,SMALLESTDRIVER

        ;Attempt to grab memory for driver.

sizeok:         mov     ah,48h
                int     21h
                jnc     grablowok

        ;Failed to grab memory - print error and exit.
```

```
                mov     dx,offset GrabLoErrMsg
                jmp     allocerr

        ;Store segment of device.

grablowok:      mov     DvcSeg,ax
                mov     BlockSize,bx

        ;Disable Ctrl-Break.

                mov     dx,offset BreakHandler
                mov     ax,2523h
                int     21h

        ;Copy interrupt vectors for later comparison.

        ;DS:TopCSeg, ES:TopCSeg
                xor     bx,bx
                mov     ds,bx
        ;DS:0000, ES:TopCSeg
                mov     si,bx
                mov     di,offset IntVectors
                mov     cx,200h
                rep     movsw

        ;Parse parameters in same way as SYSINIT does.

                mov     ds,cs:PSPSeg

        ;DS:TopPSPSeg, ES:TopCSeg

        ;Find end of filename in DI.

                mov     di,cs:NamePtr
                add     di,cs:NameLen

        ;Find end of command line in BX.

                mov     bl,byte ptr ds:[80h]
                add     bx,81h

        ;Compare them.

                cmp     bx,di
                ja      parmsgiven

        ;If they are the same, no parameters were given, so add a space.

                mov     byte ptr [di],' '
                inc     bx

        ;Append CrLf to line.

parmsgiven:     mov     word ptr [bx],0A0Dh

        ;Print 'Filename: ' if not in Quiet Mode.

                push    cs
                pop     ds

                test    ModeFlag,QuietFlag
                jnz     noprintfname

                mov     dx,offset FNameMsg
                mov     ah,9
                int     21h

        ;Make filename ASCII$ instead of ASCIIZ for printing.
```

```
                mov     di,offset NameBuffer
                mov     dx,di
                mov     cx,80h
                xor     al,al
                repnz   scasb
                dec     di
                mov     byte ptr [di],'$'

        ;Print filename.

                int     21h

        ;Restore to ASCIIZ.

                mov     byte ptr [di],0

        ;Print CrLf after filename.

                mov     dx,offset CrLfMsg
                int     21h

        ;Load driver file into memory.

        ;DS:TopCSeg, ES:TopCSeg

noprintfname:   mov     dx,offset NameBuffer
                mov     bx,offset DvcSeg
                mov     ax,4B03h
                int     21h
                jnc     loadedok

        ;Print 'EXEC failure'.

                mov     dx,offset NotLoadMsg

        ;Restore error code.

fileerr:        pop     ax

        ;Print error message and number, get offset of cause in DX.

allocerr:       call    PrintError

        ;Print final message.

prexit:         mov     ah,9
                int     21h

        ;Exit program.

exit:           mov     ax,4C00h
                int     21h

;.....................................................................

        ;DS:TopCSeg, ES:TopCSeg

        ;Check whether to print load address.

loadedok:       test    ModeFlag,VerboseFlag
                jz      noprintladdr

        ;Print 'Load address:'

                mov     dx,offset LoadAddrMsg
                mov     ah,9
                int     21h

        ;Print segment of driver.
```

```
                mov     ax,DvcSeg
                call    PHWord

        ;Print ':0000'

                mov     dx,offset Colon0Msg
                mov     ah,9
                int     21h

        ;DS:TopCSeg, ES:TopCSeg

        ;Get pointer to 'invar'

noprintladdr:   mov     ah,52h
                int     21h

        ;Store for later use.

                mov     InvarOfs,bx
                mov     InvarSeg,es

        ;DS:TopCSeg, ES:InvarSeg

        ;Fetch LastDrUsed and LastDrive from 'invar'

                mov     ax,es:[bx+20h]
                mov     word ptr LastDrUsed,ax

        ;Fetch max. sector size from 'invar'

                mov     ax,es:[bx+10h]
                mov     SecSize,ax

                push    es
                pop     ds

        ;Trace device chain from 'invar'.

        ;Point DS:DI to first block header.

                mov     si,bx

        ;Point to next block header.

notlast:        lds     si,[si]

        ;Point to pointer within block header to the next one.

                add     si,cs:word ptr NextBlHOfs

        ;Loop while not at the end of the chain.

                cmp     [si],0FFFFh
                jnz     notlast

        ;Point back at beginning of last block header in chain.

                sub     si,cs:word ptr NextBlHOfs

        ;Store for later use.

                mov     cs:ChainEndOfs,si
                mov     cs:ChainEndSeg,ds

        ;DS:DvcSeg, ES:InvarSeg

        ;Point ES:BX to device after NUL device.

                les     bx,es:[bx+22h]
```

```
        ;Point DS:SI to new device (DvcSeg:0000).

                mov     ds,cs:DvcSeg
                xor     si,si

        ;DS:DvcSeg, ES:OldDvcSeg

        ;Install all devices in chain.

anoth_dvc:      call    InstallDevice
                jnz     anoth_dvc

        ;Print LASTDRIVE error message if necessary.

                push    cs
                pop     ds

                cmp     [LDrErrMsg],'0'
                jz      noldrerr
                mov     dx,offset LDrErrMsg
                mov     ah,9
                int     21h

        ;Calculate size of driver to keep.

noldrerr:       mov     ax,BlHEndOfs
                add     ax,0Fh
                rcr     ax,1
                mov     cl,3
                shr     ax,cl

                add     ax,EndSeg

                sub     ax,DvcSeg

        ;Store size of driver to keep.

                push    ax

        ;Check whether anything installed, offer abortion if not.

                mov     ax,word ptr BlocksDone
                or      ax,ax
                jnz     somedone

        ;Nothing installed - check whether Auto Mode.

                test    ModeFlag,AutoFlag
                jnz     somedone

        ;If it didn't want to stay anyway, don't ask.

                pop     cx
                push    cx
                jcxz    somedone

        ;Check whether any INT vectors changed.

                xor     di,di
                mov     es,di

        ;DS:TopCSeg, ES:0000

                mov     si,offset IntVectors
                mov     cx,200h
                rep     cmpsw
                jnz     somedone

        ;Nothing installed, so give option of aborting.
```

```
                mov     dx,offset AskEndMsg
                mov     ah,9
                int     21h

        ;Get response from keyboard.

                mov     ah,8
badkey:         int     21h

        ;If it's an extended character code, get the second byte and try again.

                or      al,al
                jnz     realchar
                int     21h
                jmp     badkey

        ;If it's valid, act upon it, else loop for another.

realchar:       cmp     al,'N'
                jz      nokill
                cmp     al,'n'
                jz      nokill
                cmp     al,'Y'
                jz      kill
                cmp     al,'y'
                jnz     badkey

        ;Response was yes, so set length required to zero.

kill:           pop     bx
                xor     bx,bx
                push    bx

                ;mov     bx,DvcSeg
                ;mov     EndSeg,bx
                ;xor     bx,bx
                ;mov     BlHEndOfs,bx


        ;Print the key pressed.

nokill:         mov     ah,02h
                mov     dl,al
                int     21h

        ;Print CrLf afterwards.

                mov     dx,offset CrLfMsg
                mov     ah,9
                int     21h

        ;Print Lastdrive and LastDrUsed if blocks done and Verbose Mode.

somedone:       push    cs
                pop     ds

                test    BlocksDone,0FFh
                jz      noprintldrmsg

                test    ModeFlag,VerboseFlag
                jz      noprintldrmsg

                mov     ax,word ptr LastDrUsed

                mov     LDMsgA,'A'-1
                mov     LDMsgB,'A'-1

                add     byte ptr LDMsgA,al
                add     byte ptr LDMsgB,ah
```

```
                mov     dx,offset LastDrMsg
                mov     ah,9
                int     21h

        ;If not Quiet Mode, print number of devices installed.

noprintldrmsg:  test    ModeFlag,QuietFlag
                jnz     noprintnuminst

        ;Get driver keep size into CX, don't print installed message if zero.

                pop     cx
                push    cx
                jcxz    noprintnuminst

        ;Print number of blocks installed, if any.

                mov     bl,BlocksDone
                or      bl,bl
                jz      noblocks

                add     NumBlInstMsg,bl
                mov     dx,offset NumBlInstMsg
                mov     ah,9
                int     21h

                mov     dx,offset NumInstMsgA
                cmp     bl,1
                jnz     blnoplural
                inc     dx
blnoplural:     int     21h

        ;Print number of character devices installed, if any.

noblocks:       mov     bl,CharsDone
                or      bl,bl
                jz      noprintnuminst

                add     NumChInstMsg,bl
                mov     dx,offset NumChInstMsg
                mov     ah,9
                int     21h

                mov     dx,offset NumInstMsgA
                cmp     bl,1
                jnz     chnoplural
                inc     dx
chnoplural:     int     21h

        ;Insert new LastDrUsed into 'invar'.

noprintnuminst: les     bx,Invar

                mov     al,LastDrUsed
                mov     byte ptr es:[bx+20h],al

        ;Restore driver size in paragraphs.

                pop     bx

        ;Test whether to print driver size.

                test    ModeFlag,VerboseFlag
                jz      noprintsize

        ;Print 'Size of driver in paras:'.

                mov     dx,offset SizeMsg
                mov     ah,9
                int     21h
```

```
                mov     ax,bx
                call    PHWord

                mov     dx,offset CrLfMsg
                mov     ah,9
                int     21h

noprintsize:    or      bx,bx
                jnz     lengthnotzero

        ;Length of driver is zero, so exit now.

                jmp     exit

        ;Check whether it fits in the allocated block.

lengthnotzero:  cmp     ax,BlockSize
                jna     drvrfits
                mov     dx,offset TooBigMsg
                mov     ah,9
                int     21h

        ;Change memory allocation on lowest block in memory.

drvrfits:       mov     es,DvcSeg
                mov     ah,4Ah
                int     21h
                jnc     allocok

        ;Print 'allocation error'.

                mov     dx,offset Reduce2ErrMsg
                call    PrintError
                mov     ah,9
                int     21h
                mov     dx,offset Reduce2ErrMsga
                mov     ah,9
                int     21h

        ;Check whether to print INT vectors changed.

allocok:        test    ModeFlag,VerboseFlag
                jz      noprintints

        ;Print 'Interrupt vectors changed:'

                mov     dx,offset IntChangeMsg
                mov     ah,9
                int     21h

        ;Print all INTs changed.

                xor     bx,bx
                mov     es,bx
                mov     di,bx
                mov     si,offset IntVectors
                mov     cx,200h

        ;Print no comma before the first INT number.

                mov     dx,offset CommaMsg-1

        ;Compare INT vectors with copy taken earlier.

loopints:       rep     cmpsw

        ;If we stopped at the end, leave the loop.

                jcxz    lastdiff
```

```
        ;Flag that at least one was changed.

                or      bl,1

        ;Print 'h, ' between INT numbers.

                mov     ah,9
                int     21h

        ;Check whether it was offset or segment that was different.

                mov     ax,di
                dec     ax
                test    ax,2
                jnz     notofs

        ;If was the offset, so don't bother checking the segment.

                add     di,2
                add     si,2
                dec     cx

        ;Print interrupt number.

notofs:         shr     ax,1
                shr     ax,1
                call    PHByte

        ;After the first one, print 'h, ' before the rest.

                mov     dx,offset CommaMsg
                jmp     loopints

        ;Print either full stop or 'None.'

lastdiff:       mov     dx,offset FullStopMsg

        ;Test flag to see whether any were changed.

                or      bl,bl
                jnz     notnochanges

        ;None were changed, so print 'None.' instead of a full stop.

                mov     dx,offset NoneMsg

notnochanges:   mov     ah,9
                int     21h

        ;DS:TopCSeg, ES:0000

        ;Link from NUL device.

noprintints:    les     bx,Invar
                add     bx,0022h
                mov     ax,NewDrvOfs
                mov     es:[bx],ax
                mov     ax,NewDrvSeg
                mov     es:[bx+2],ax

                push    cs
                pop     es

        ;DS:TopCSeg, ES:TopCSeg

        ;Find last backslash in filename.

                mov     di,offset NameBuffer+80h
                mov     al,'\'
```

```
                std
                mov     cx,0080h

                repnz   scasb
                cld
                add     di,2

        ;Point to arena header of driver segment.

                mov     ax,DvcSeg
                dec     ax
                mov     es,ax

        ;Set driver segment to self-ownership.

                inc     ax
                mov     word ptr es:[1],ax

        ;DS:TopCSeg, ES:DvcSeg-1

        ;Move name into arena header.

                mov     si,di
                mov     di,8
                mov     cx,8
movname:        lodsb
                cmp     al,2eh
                jz      fill0s
                cmp     al,0
                jz      fill0s
                stosb
                loop    movname
                jmp     stayexit

fill0s:         xor     al,al
                rep     stosb

stayexit:       mov     dx,offset StayingMsg
                jmp     prexit



;.....................break handler.....................................

BreakHandler:   iret                    ;well, that didn't take long!

;.....................PHWord.........................

;       IN:     AX      word to be printed
;       OUT:    nothing
;       LOST:   nothing


PHWord:         push    ax
                xchg    al,ah
                call    PHByte
                mov     al,ah
                call    PHByte
                pop     ax
                ret

;................PHByte...........

;       IN      AL      byte to printed
;       OUT:    nothing
;       LOST:   nothing

PHByte:         push    ax
                mov     ah,al
```

```
                shr     al,1
                shr     al,1
                shr     al,1
                shr     al,1
                call    PHNibble
                mov     al,ah
                call    PHNibble
                pop     ax
                ret

;..............PHNibble.................

;       IN:     AL      nibble to be printed
;       OUT:    nothing
;       LOST:   nothing

PHNibble:       push    dx
                push    ax
                and     al,0Fh
                add     al,'0'
                cmp     al,'9'
                jna     ph1
                add     al,'A'-'9'-1
ph1:            mov     ah,02h
                mov     dl,al
                int     21h
                pop     ax
                pop     dx
                ret


;.....................PrintError.....................................

;       IN:     AX      Error message to explain.
;               DX      Offset of error message.
;               DS      CSeg
;       OUT:    DX      Offset of error cause message.
;       LOST:   AX
;               BX

        ;Print first message.

PrintError:     push    ax
                mov     ah,9
                int     21h
                pop     ax

        ;Print error number.

                call    PHWord

        ;If over 0Bh, zero it - (unknown).

                cmp     ax,000Bh
                jb      notoverB
                xor     ax,ax

        ;Look up offset of error message.

notoverB:       mov     bx,offset ErrTable
                shl     ax,1
                add     bx,ax
                mov     dx,[bx]
                ret


;.........................InstallDevice.............................

;       IN:     DS:SI   address of new driver header
;               ES:BX   address of old driver header
```

```
;       OUT:    DS:SI   address of next driver header
;               ES:BX   address of new driver header
;               ZERO    set if last driver in file


        ;Store device addresses.

InstallDevice:  mov     cs:NewDrvOfs,si
                mov     cs:NewDrvSeg,ds
                mov     cs:OldDrvOfs,bx
                mov     cs:OldDrvSeg,es

                push    cs
                pop     ds

        ;Print CrLf to keep display tidy.

        ;DS:TopCSeg, ES:OldDvcSeg

                mov     dx,offset CrLfMsg
                mov     ah,9
                int     21h

        ;Set up request header.

        ;Insert next block device number.

                mov     al,LastDrUsed
                mov     byte ptr [RqHdr+16h],al

        ;Insert command number zero - INIT.

                mov     byte ptr [RqHdr+2],0

        ;Insert default end of driver address = start of driver.

                mov     ax,DvcSeg
                mov     word ptr [RqHdr+10h],ax
                mov     word ptr [RqHdr+0Eh],0

        ;Insert default no blocks in driver.

                mov     byte ptr [RqHdr+0Dh],0

        ;Insert pointer to copy of command line.

                mov     ax,PSPSeg
                mov     word ptr [RqHdr+14h],ax
                mov     ax,NamePtr
                mov     word ptr [RqHdr+12h],ax

                push    cs
                pop     es

        ;DS:TopCSeg, ES:TopCSeg

        ;Store registers (don't count on driver to keep them).

                push    si

        ;Set ES:BX to point to RqHdr (for DvcStrat call.)

                mov     bx,offset RqHdr

        ;Set up return addresses on stack.

                mov     ds,cs:NewDrvSeg

        ;DS:NewDrvSeg, ES:TopCSeg
```

```
        ;Push far address of DEVLOAD.

                push    cs
                mov     ax,offset after_int
                push    ax

        ;Push far address of dvc_int.

                push    ds
                push    word ptr ds:[si+8]

        ;Push far address of dvc_strat

                push    ds
                push    word ptr ds:[si+6]

        ;Pass control to dvc_strat, which RETFs to dvc_int, which
        ;in turn RETFs to after_int.

BREAKPOINT2:    retf

        ;Restore registers.

after_int:      pop     si

        ;Increase count of character devices if it is one.

                test    byte ptr ds:[si+5],80h
                push    cs
                pop     ds
                jz      notchardev
                inc     CharsDone

        ;Print CrLf after driver.

notchardev:     mov     dx,offset CrLfMsg
                mov     ah,9
                int     21h

        ;Get offset of end of driver.

                mov     ax,word ptr ds:[RqHdr+0Eh]

        ;Convert to paragraphs (rounded up.)

                add     ax,0Fh
                rcr     ax,1
                mov     cl,3
                shr     ax,cl

        ;Add segment of end of driver.

                add     ax,word ptr ds:[RqHdr+10h]

        ;Compare with previous value of EndSeg

                cmp     ax,EndSeg
                jb      endset

        ;EndSeg has increased - this is only a problem if blocks are
        ;already installed.

                test    BlocksDone,0FFh
                jz      oktogrow

        ;Some block headers are already at EndSeg - can't change it now.

                mov     dx,offset BadIncMsg
                mov     ah,9
```

```
                int     21h
                jmp     endset

        ;No block headers done yet - change EndSeg.

oktogrow:       mov     EndSeg,ax

        ;DS:TopCSeg

        ;Check number of units in driver. Skip if none.

endset:         mov     ch,[RqHdr+0Dh]
                or      ch,ch
                jnz     yesunits

        ;No units in this device, so skip the next section.

                jmp     nounits

        ;Zero count of block number in this device.

yesunits:       xor     cl,cl

        ;Point ES:BP to new block header location.

                les     bp,BlHEnd

        ;Point DS:BX to BPB pointer array from driver.

                lds     bx,dword ptr [RqHdr+12h]

        ;Point DS:SI to next BPB and increase pointer.

nextblk:        mov     si,[bx]
                inc     bx
                inc     bx

        ;Check sector size.

                mov     ax,[si]
                cmp     ax,cs:[SecSize]
                jna     secsizeok
                jmp     secsizeerr

secsizeok:      mov     al,cs:LastDrUsed
                mov     cs:[BlHdrMsgA],'A'
                add     cs:[BlHdrMsgA],al

        ;Check lastdrive.

                cmp     al,cs:LastDrive
                jnz     ldrok
                jmp     ldrerr

        ;Increase LastDrUsed.

ldrok:          inc     cs:LastDrUsed

        ;Store absolute block no. and block no. in device.

                mov     ah,cl
                inc     cl
                mov     es:[bp],ax

        ;Store pointer to BPB ptr array.

                push    ds
                push    bx

        ;Point DS:BX to last block header in chain.
```

```
                lds     bx,cs:ChainEnd

        ;Make it point to the new one.

                add     bx,cs:word ptr NextBlHOfs
                mov     [bx],bp
                mov     [bx+2],es
                sub     bx,cs:word ptr NextBlHOfs

                push    cs
                pop     ds

        ;Store new pointer to end of block header chain.

                mov     ChainEndOfs,bp
                mov     ChainEndSeg,es

        ;Print address of new block header if Verbose Mode.

                test    ModeFlag,VerboseFlag
                jz      noprintblhmsg

                mov     dx,offset BlHdrMsg
                mov     ah,9
                int     21h

                mov     ax,es
                call    PHWord
                mov     ah,02h
                mov     dl,3ah
                int     21h
                mov     ax,bp
                call    PHWord

                mov     dx,offset CrLfMsg
                mov     ah,9
                int     21h

        ;Get pointer to LASTDRIVE array.

noprintblhmsg:  lds     bx,Invar
                lds     bx,[bx+16h]

        ;Calculate offset in array of entry for this drive.

                mov     al,cs:LastDrUsed
                dec     al
                mov     ah,cs:LDrSize
                mul     ah
                add     bx,ax

        ;Insert 'valid' flag.

                mov     byte ptr [bx+44h],40h

        ;Insert pointer to block header for this drive.

                mov     word ptr [bx+45h],bp
                mov     word ptr [bx+47h],es

        ;Restore pointer to BPB pointer array.

                pop     bx
                pop     ds

        ;Insert pointer to device into block header.

                mov     ax,cs:NewDrvOfs
                add     bp,cs:word ptr NextBlHOfs
```

```
                mov     es:[bp-6],ax
                mov     ax,cs:NewDrvSeg
                mov     es:[bp-4],ax

        ;Insert 'BPB needs rebuilding' flag into block header.

                mov     byte ptr es:[bp-1],0FFh

        ;Insert 'End of chain' into block header.

                mov     es:[bp],0FFFFh
                sub     bp,cs:word ptr NextBlHOfs

        ;Expand BPB into block header.

                mov     ah,53h                  ;hello woody y doesnt this work
                int     21h

        ;Point ES:BP to location of next block header.

                add     bp,cs:BlHSize

        ;Store location of next block header.

                mov     cs:BlHEndOfs,bp

        ;Increase count of blocks installed.

                inc     cs:BlocksDone

        ;Loop if more blocks in this device to install.

nxtblkchk:      cmp     cl,ch
                jz      nounits
                jmp     nextblk

        ;Sector size too big - print error and fail to install this block.

secsizeerr:     push    cs
                pop     ds
                mov     dx,offset SSizeErrMsg
                mov     ah,9
                int     21h
                inc     cl
                jmp     nxtblkchk

        ;Lastdrive too small - signal error and don't install any more.

ldrerr:         push    cs
                pop     ds
                sub     ch,cl
                add     [LDrErrMsg],ch

        ;Finished installing units, or was none to install.

nounits:        push    cs
                pop     ds

        ;Print init return status if not Verbose Mode.

                test    ModeFlag,VerboseFlag
                jz      noprintinitret

                mov     dx,offset InitRetMsg
                mov     ah,9
                int     21h
                mov     ax,word ptr [RqHdr+3]
                call    PHWord

                mov     dx,offset CrLfMsg
```

```
                mov     ah,9
                int     21h

        ;DS:TopCSeg

        ;Set up pointers.

noprintinitret: les     bx,OldDrv
                lds     si,NewDrv

        ;DS:NewDvcSeg, ES:OldDvcSeg

        ;Find location of next driver in file.

                mov     ax,ds:[si+2]
                push    ax

                mov     ax,ds:[si]
                push    ax

        ;Link NewDrv to OldDrv.

                mov     ds:[si],bx
                mov     ds:[si+2],es

        ;Restore next driver in file address to AX:SI

                pop     si
                pop     ax

        ;Point ES:BX to just installed driver.

                les     bx,cs:NewDrv

        ;Check whether to change segment or use same seg for next driver.

                cmp     ax,0FFFFh
                jz      nosegchange

        ;Segment is different - add value onto old segment and put into DS.

                mov     cx,ds
                add     cx,ax
                mov     ds,cx

        ;Set zero flag on whether this is the last driver in the file.

nosegchange:    cmp     si,0FFFFh
                ret

;.............................TEXT MESSAGES.................................

StayingMsg      db 'Driver staying resident.',13,10,24h
FNameMsg        db 'Filename     : $'
LoadAddrMsg     db 'Load address : $'
InitRetMsg      db 'Init function return status : $'
SizeMsg         db 'Size of driver (paragraphs) : $'
IntChangeMsg    db 'Interrupt vectors changed   : $'
NoneMsg         db 'None.',13,10,24h
CommaMsg        db 'h, $'
FullStopMsg     db 'h.',13,10,24h
Colon0Msg       db ':'
NotStayMsg      db '0000'
CrLfMsg         db 13,10,24h
LastDrMsg       db 13,10,'Last drive in use : '
LDMsgA          db 'A:',13,10,'Last drive avail. : '
LDMsgB          db 'A:',13,10,24h
BlHdrMsg        db 'Block header for drive '
BlHdrMsgA       db 'A: at $'
NumBlInstMsg    db '0 block$'
```

```
NumChInstMsg    db '0 character device$'
NumInstMsgA     db 's installed.',13,10,24h
LDrErrMsg       db '0 block(s) not installed - '
                db 'LASTDRIVE= parameter in CONFIG.SYS too small.',13,10,24h
AskEndMsg       db 13,10,'No blocks or INTs installed - terminate (Y/N) ? $'
SSizeErrMsg     db 'Block not installed - sector size too large.',13,10,24h

        ;Error messages.

ReduceErrMsg    db "Error: Can't reduce memory allocation ($"
RelEnvErrMsg    db "Error: Can't release environment ($"
RelPSPErrMsg    db "Error: Can't release original segment ($"
GrabHiErrMsg    db "Error: Can't grab enough memory to relocate ($"
GrabLoErrMsg    db "Error: Can't grab memory to load driver ($"
Reduce2ErrMsg   db 13,10,"Error: Can't change final memory allocation ($"
Reduce2ErrMsga  db " Have installed driver; continuing anyway.",13,10
                db " Rebooting your system is recommended.",13,10,10,24h
BadIncMsg       db 'Error: End of driver address returned has increased after'
                db 13,10,'       block header(s) installed.',13,10
                db '       Increased request will be ignored.',13,10,24h
NotLoadMsg      db 13,10,'Error: EXEC failed ($'
TooBigMsg       db 13,10,'Error: Driver requested more memory than is '
                db 'available.',13,10,24h

        ;Error cause messages.

Err2            db 'h - File not found)',13,10,24h
Err3            db "h - Directory doesn't exist)",13,10,24h
Err5            db 'h - Access denied)',13,10,24h
Err7            db 'h - Arena header corrupted)',13,10,24h
Err8            db 'h - Out of memory)',13,10,24h
Err9            db 'h - Wrong segment passed!)',13,10
                db ' PLEASE INFORM THE AUTHOR!',13,10,24h
ErrB            db 'h - Format invalid)',13,10,24h
ErrUnknown      db 'h'
BadSwitchMsg2   db ')',13,10,24h


ErrTable        dw      ErrUnknown
                dw      ErrUnknown
                dw      Err2
                dw      Err3
                dw      ErrUnknown
                dw      Err5
                dw      ErrUnknown
                dw      Err7
                dw      Err8
                dw      Err9
                dw      ErrUnknown
                dw      ErrB

;.............................PROGRAM DATA.................................

LDrSize         db      58h,0   ;size of block in LastDrive array.
BlHSize         dw      0021h   ;size in paras of parameter block.
NextBlHOfs      db      19h,0   ;offset in parameter block of ptr to next one.

BlocksDone      db      00              ;no. of blocks installed.
CharsDone       db      00              ;no. of character devices installed.

DvcSeg          dw      ?               ;parameter block for EXEC function.
                dw      0000            ;i.e. segment, relocation factor.

ModeFlag        db      00

BlHEnd          label   dword
BlHEndOfs       dw      0000
EndSeg          dw      ?               ;segment, end of required memory.

PSPSeg          dw      ?
```

```
PathPtr         dd      0

NameBuffer      db      80h dup(?)

LastDrUsed      db      ?
LastDrive       db      ?

OldAllocStrat   dw      ?
BlockSize       dw      ?

NamePtr         dw      ?               ;pointer to start of name.
NameLen         dw      ?

Invar           label   dword
InvarOfs        dw      ?
InvarSeg        dw      ?

ChainEnd        label   dword
ChainEndOfs     dw      ?               ;last device parameter block in chain.
ChainEndSeg     dw      ?

SecSize         dw      ?

NewDrv          label   dword
NewDrvOfs       dw      ?               ;storage for InstallDevice routine.
NewDrvSeg       dw      ?

OldDrv          label   dword
OldDrvOfs       dw      ?
OldDrvSeg       dw      ?

RqHdr           db      20h dup (?)

IntVectors      dw      200h dup (?)    ;storage for interrupt vectors,
                                        ;for checking whether they've changed.

        ;Marker to signal last byte that needs relocation.

LASTBYTE        equ     $

;.................DATA WHICH ISN'T NEEDED AFTER RELOCATION..................

SignOnMsg       db 'DEVLOAD.EXE v3.0  (C) 1992, 1993 David Woodhouse.',13,10
                db ' Loads device drivers from the command line.',13,10,10,24h

HelpMsg1        db 'Usage: DEVLOAD [<switches>] <filename> [<params>]',13,10
                db 'Emulates device=<filename> [<params>] in CONFIG.SYS',13,10
                db 10,'Switches:',13,10
                db '      /A - automatic mode (no abortion).',13,10
                db '   /? /H - display this help message.',13,10
                db '      /Q - quiet mode.',13,10
                db '      /V - verbose mode.',13,10
                db 10,'Not supported by DR-DOS or MS-DOS before 3.00',13,10,10
                db 'Breakpoints for debugging drivers -',13,10
                db 'Relocation RETF : $'
HelpMsg2        db 13,10,'Execution  RETF : $'


BadSwitchMsg    db 'Error: Bad switch ($'
NoFileMsg       db 'Error: No filename given. Use DEVLOAD /? for instructions.'
                db 13,10,24h
FileNoExistMsg  db "Error: Can't find file ($"
BadVerMsg       db 'Error: This program uses version-specific information, and'
                db 13,10,'       only supports MS-DOS versions 3 to 6.'
                db 13,10,24h


;.........MAIN PROGRAM ENTRY POINT - SITUATE ABOVE LASTBYTE BECAUSE.........
;......IT DOESN'T NEED TO BE KEPT WHEN RELOCATING TO THE TOP OF MEMORY.......
```

```
        ;Set up segment registers.

Main:           push    cs
                pop     ds
                push    cs
                pop     es
                cld

        ;Get PSP segment.

                mov     ah,62h
                int     21h
                mov     cs:PSPSeg,bx

        ;Print sign on message.

                mov     dx,offset SignOnMsg
                mov     ah,9
                int     21h

        ;Check DOS version.

                mov     ax,3000h
                int     21h
                cmp     al,3
                ja      okver
                jz      ver3

        ;Version before 3.0, so print error and exit.

                mov     dx,offset BadVerMsg
                jmp     prexit

        ;Version 3.x, so change variables to correct values.

ver3:           mov     LDrSize,51h
                mov     byte ptr BlHSize,20h
                mov     NextBlHOfs,18h

        ;Check command line.

okver:          mov     ds,PSPSeg
                xor     bh,bh
                mov     bl,byte ptr ds:[80h]
                or      bx,bx
                jnz     cmdlineexists

        ;No parameters given - print error and exit.

nofilename:     mov     dx,offset NoFileMsg
                push    cs
                pop     ds
                jmp     prexit

        ;Command line exists - convert to all upper case.

cmdlineexists:  mov     si,0081h
                mov     cx,bx
toupperloop:    lodsb
                cmp     al,'a'
                jb      notlower
                cmp     al,'z'
                ja      notlower
                xor     al,20h
                mov     [si-1],al
notlower:       loop    toupperloop

        ;Check whether filename present.
```

```
                mov     si,0081h
                add     bx,si

        ;DS:SI--> Start of command line.
        ;DS:BX--> End of command line.

getloop1:       lodsb

        ;If passed end of command line, exit loop.

                cmp     si,bx
                ja      nofilename

        ;Loop while whitespace.

                cmp     al,' '
                jz      getloop1
                cmp     al,9
                jz      getloop1

        ;Found non-whitespace, point back at it.

                dec     si

        ;DS:SI --> first non-whitespace char on command line.

        ;Get current switch char (usually '/').

                mov     ax,3700h
                int     21h

        ;Check whether first char on command line is a switch.

                cmp     [si],dl
                jz      isswitch
                jmp     noswitch

        ;Load switch and check it.

isswitch:       lodsw
                cmp     ah,'?'
                jz      help
                cmp     ah,'H'
                jz      help
                cmp     ah,'Q'
                jz      quiet
                cmp     ah,'A'
                jz      auto
                cmp     ah,'V'
                jz      verbose

        ;Unrecognised switch - print error and exit.

unknownswitch:  push    ax

                push    cs
                pop     ds

                mov     dx,offset BadSwitchMsg
                mov     ah,9
                int     21h

                pop     dx
                mov     ah,2
                int     21h
                mov     dl,dh
                int     21h

                mov     dx,offset BadSwitchMsg2
                jmp     prexit
```

```
        ;Print help message.

help:           push    cs
                pop     ds

                mov     dx,offset HelpMsg1
                mov     ah,9
                int     21h
                mov     ax,offset BREAKPOINT1
                call    PHWord
                mov     dx,offset HelpMsg2
                mov     ah,9
                int     21h
                mov     ax,offset BREAKPOINT2
                call    PHWord
                mov     dx,offset CrLfMsg
                jmp     prexit

        ;Set verbose mode flag.

verbose:        or      cs:ModeFlag,VerboseFlag
                and     cs:ModeFlag,not QuietFlag
                jmp     switchloop

        ;Set automatic mode flag.

auto:           or      cs:ModeFlag,AutoFlag
                jmp     switchloop

        ;Set quiet mode flag.

quiet:          or      cs:ModeFlag,QuietFlag
                and     cs:ModeFlag,not VerboseFlag

        ;Skip to next space.

switchloop:     lodsb
                cmp     si,bx
                jna     switchloop1
                jmp     nofilename

switchloop1:    cmp     al,9
                jz      outswitchloop
                cmp     al,' '
                jnz     switchloop

        ;Point back at first space and go back to getloop1 to skip
        ;to either next switch or to filename.

outswitchloop:  dec     si
                jmp     getloop1

        ;Store pointer to start of pathname.

noswitch:       push    si
                mov     bp,si

        ;Find pointer to actual 8-char filename and end of pathname.

getloop2:       lodsb
                cmp     al,'\'
                jz      backsl
                cmp     al,'/'
                jnz     nobacksl

        ;Move pointer to after backslash into BP.

backsl:         mov     bp,si
```

```
        ;Break out of loop if space, tab, CR or LF found.

nobacksl:       cmp     al,' '
                jz      outloop2
                cmp     al,9
                jz      outloop2
                cmp     al,13
                jz      outloop2
                cmp     al,10
                jz      outloop2

        ;Check whether end of command line reached. Loop if not.

                cmp     si,bx
                jna     getloop2

        ;DS:SI-2 --> last char of pathname.
        ;DS:BP --> first char in filename.

        ;Calculate length of filename.

outloop2:       mov     es:NamePtr,bp
                dec     si
                mov     cx,si
                sub     si,bp
                mov     es:NameLen,si

        ;Restore pointer to start of pathname.

                pop     si

        ;Check whether file specified contains path.

                cmp     si,bp
                jz      notpathname

        ;Set PathPtr to point to zero - simulate no PATHs left.

                mov     word ptr es:PathPtr,offset DvcSeg+2
                mov     word ptr es:PathPtr+2,cs

        ;Start with default directory.

notpathname:    sub     cx,si
                mov     di,offset NameBuffer

        ;Use default directory or one specified first time, not PATH.

                jmp     entrypoint

;.....................................................................

        ;Filename doesn't exist as specified - try using PATH.

        ;Check whether we've already got a pointer to PATH.

allpathloop:    lds     si,PathPtr
                or      si,si
                jnz     pathfound

        ;Not yet, so find PATH segment.

                mov     ds,cs:PSPSeg
                mov     bx,word ptr ds:[002Ch]

        ;Check whether it exists.

                or      bx,bx
                jnz     envsegexists
```

```
        ;No more PATH items or no PATH segment, so print error and exit.

filenoexist:    push    cs
                pop     ds
                mov     dx,offset FileNoExistMsg
                jmp     fileerr

        ;Store PATH segment in local pointer.

envsegexists:   mov     ds,bx
                mov     word ptr cs:PathPtr+2,bx

        ;Scan environment for 'PATH='

envloop1:       lodsb
                cmp     al,0
                jz      filenoexist
                cmp     al,'P'
                jnz     nextenvvar1
                lodsb
                cmp     al,'A'
                jnz     nextenvvar1
                lodsb
                cmp     al,'T'
                jnz     nextenvvar1
                lodsb
                cmp     al,'H'
                jnz     nextenvvar1
                lodsb
                cmp     al,'='
                jnz     nextenvvar1
                jmp     pathfound

        ;Not 'PATH=', so skip to next environment variable.

nextenvvar:     lodsb
nextenvvar1:    or      al,al
                jnz     nextenvvar
                jmp     envloop1

        ;Store file error message.

pathfound:      push    ax

        ;Skip spaces at start of this PATH item.

pathfoundloop:  lodsb
                cmp     al,' '
                jz      pathfoundloop
                cmp     al,9
                jz      pathfoundloop

        ;DS:SI-1 --> first non-whitespace in PATH item.

        ;If we've reached the end of the PATH statement, error and exit.

                cmp     al,0
                jz      filenoexist

        ;Forget file error message - we'll try again.

                add     sp,2

        ;Store start of this PATH item + 1.

                push    si

        ;Find end of this PATH item.

pathloop1:      lodsb
```

```
                cmp     al,0
                jz      endpath
                cmp     al,';'
                jnz     pathloop1

        ;Store start of next PATH item.

endpath:        mov     word ptr cs:PathPtr,si

        ;If last one, point back at the terminating NULL.

                or      al,al
                jnz     ismorepaths
                dec     word ptr cs:PathPtr

        ;Calculate length of this PATH item.

ismorepaths:    mov     cx,si
                pop     si
                sub     cx,si
                dec     si

        ;Copy PATH item to NameBuffer.

                mov     di,offset NameBuffer
                rep     movsb

        ;Add backslash if necessary.

                push    cs
                pop     ds

                cmp     byte ptr [di-1],'\'
                jz      alreadybacksl
                mov     al,'\'
                stosb

        ;Copy filename after PATH item.

alreadybacksl:  mov     si,NamePtr
                mov     cx,NameLen

                mov     ds,PSPSeg

entrypoint:     rep     movsb

        ;Store terminating NULL.

                xor     al,al
                stosb

        ;Check whether file exists by attempting to get attributes.

                push    cs
                pop     ds

                mov     dx,offset NameBuffer
                mov     ax,4300h
                int     21h
                jnc     okfilename
                jmp     allpathloop

        ;File exists - expand filename using function 60h.

okfilename:     mov     si,offset NameBuffer
                mov     di,si
                mov     ah,60h
                int     21h

        ;Get old allocation strategy.
```

```
                mov     ax,5800h
                int     21h
                mov     OldAllocStrat,ax

        ;Reduce main allocation.

        ;DS:CSeg, ES:CSeg

                mov     bx,offset LASTBYTE+10Fh
                add     bx,offset STACKLEN
                mov     cl,4
                shr     bx,cl
                mov     cx,bx
                mov     es,PSPSeg
                mov     ah,4ah
                int     21h
                jnc     reduceok

        ;Failed to reduce memory allocation, so print error and exit.

                mov     dx,offset ReduceErrMsg
                jmp     allocerr

        ;Set allocation strategy to highest fit.

reduceok:       mov     ax,5801h
                mov     bx,2
                int     21h

        ;Request enough at top of mem for PSP + DEVLOAD + STACK.

                mov     bx,cx
                mov     ah,48h
                int     21h
                pushf
                mov     es,ax

        ;Reset allocation strategy to old value.

        ;DS:CSeg, ES:TopPSPSeg

                mov     ax,5801h
                mov     bx,OldAllocStrat
                int     21h

        ;Check whether grabbed memory OK.

                popf
                jnc     nograbhierr

        ;Failed to grab memory, so print error and exit.

                mov     dx,offset GrabHiErrMsg
                jmp     allocerr

        ;Make new PSP at top of memory.

        ;DS:CSeg, ES:TopPSPSeg

nograbhierr:    mov     ds,PSPSeg
                mov     si,word ptr [2]
                mov     dx,es
                mov     ah,55h
                int     21h

        ;Fix parent PSP record in new PSP.

                mov     ax,ds:[16h]
                mov     es:[16h],ax
```

```
        ;Move program to top of memory.

                mov     cx,offset LASTBYTE+8Fh
                and     cx,0FFF0h
                shr     cx,1
                mov     di,80h
                mov     si,di
                rep     movsw

        ;Make segment at top of memory self-owned.

                push    cs
                pop     ds

                mov     ax,es
                dec     ax
                mov     ds,ax
                mov     word ptr ds:[1],es

                push    cs
                pop     ds

        ;Calculate location of stack at top of memory.

                mov     bx,offset LASTBYTE+10Fh
                mov     cl,4
                shr     bx,cl
                mov     ax,es
                add     ax,bx

        ;Change to stack at top of memory.

                mov     ss,ax

        ;Make PSP at top of memory current.

                mov     bx,es
                mov     ah,50h
                int     21h

        ;Transfer control to top of memory via RETF.

                mov     ax,es
                add     ax,10h
                mov     ds,ax
                push    ax
                mov     ax,offset relocated
                push    ax
BREAKPOINT1:    retf




CSeg            ends

SSeg    segment stack   para    'STACK'

        org     0

        db      STACKLEN dup (?)


SSeg            ends

        end     Main
```

APPENDIX 5 - NETLIST SOURCE CODE.

```
#include <stdio.h>
#include <string.h>
#include <dos.h>

void Disconnect (char *devname);
void ShowConnections();
void ShowUse();
void Connect (char *devname, char *extname, char *password);

struct REGPACK regs;
char localname[16];
char netname[128];

void main(int argc, char **argv)
  {

  printf ("NETLIST.EXE v1.2 (C) 1992 David Woodhouse.\n\n");

  regs.r_ax=0x5e00;  /* get local machine name */
  regs.r_dx=(unsigned)netname;
  regs.r_ds=FP_SEG(netname);

  intr (0x21,&regs);

  if (regs.r_flags&1) /* if carry set */
    printf ("\007Error: MS-Net not running on this system.\n");
  else
    {
    if (argc<2)
      ShowConnections();
    else
    if (argc<3 || !strcmp (argv[1],"/?"))
      {
      ShowUse();
      ShowConnections();
      }
    else
    if (!stricmp (argv[2],"/d"))
      Disconnect (argv[1]);
    else
    if (!stricmp (argv[1],"/d"))
      Disconnect (argv[2]);
    else
    Connect (argv[1],argv[2],argv[3]);

    }

  }


/*        ShowConnections() - Show all net redirections        */

void ShowConnections()
  {
  int count=0;

  printf ("Local machine name: %s\n\n"
        "Network connections:\n      Local name    Network name\n"
        ,netname);

  regs.r_ax=0x5f02;               /* get redirection entry #BX */
  regs.r_es=FP_SEG(netname);
  regs.r_di=(unsigned)netname;
  regs.r_si=(unsigned)localname;
  regs.r_bx=0;
```

```
  intr (0x21,&regs);

  do
    {
    printf ("%16s %s\n",localname,netname);
    regs.r_ax=0x5f02;
    regs.r_bx=++count;
    intr (0x21, &regs);
    }
  while (!(regs.r_flags&1));      /* while carry not set */

  }

void ShowUse()
  {
  printf (
   "Use: NETLIST <localname> /D                       Disconnect device\n"
   "     NETLIST <localname> <netname> [<password>]  Attach device\n"
   "     NETLIST                                     List net devices\n"
   "     NETLIST /?                                  Display this list\n\n");
  }

void Disconnect(char *devname)
  {
  regs.r_ax=0x5f04;
  regs.r_si=(unsigned)devname;
  regs.r_ds=FP_SEG(devname);

  strupr (devname);

  intr (0x21,&regs);
  if (regs.r_flags&1)
    {
    /* carry set - error */
    printf (" Error disconnecting %s ",devname);

    switch (regs.r_ax)     /* holds error code */
      {
      case 15:
        printf ("Device not connected.\n");
        break;

      default:
        printf ("No. %2X\n",regs.r_ax);
      }
    }
  else
    {
    if (devname[1]==':')
      printf ("Drive");
    else
      printf ("Device");
    printf(" %s disconnected.\n",devname);
    }
  }

void Connect (char *devname, char *extname, char *password)
  {
  strupr (devname);
  strupr (extname);

  if (password==NULL)
    password="\0";
  else
    strupr (password);

  regs.r_bx=(devname[1]==':')?4:3;    /* set printer/drive flag in BL */
```

```
regs.r_cx=3;       /* arbitrary parameter - why not 3? */
regs.r_ax=0x5f03;
regs.r_ds=FP_SEG(devname);
regs.r_si=(unsigned)devname;
regs.r_es=FP_SEG(netname);
regs.r_di=(unsigned)netname;

sprintf(netname, "%s%c%s",extname, '\0', password);

intr (0x21, &regs);
if (regs.r_flags&1)
  {
  /* carry set - error */
  printf (" Error connecting %s to %s -\n",devname, extname);

  switch (regs.r_ax)
    {
    case 0x35:
      printf ("Network path not found.\n");
      break;

    case 0x56:
      printf ("Access denied.\n");
      break;

    case 0x47:
      printf ("Server closed.\n");
      break;

    case 0x55:
      if (devname[1]==':')
        printf ("Drive");
      else
        printf ("Device");
      printf (" in use.\n");
      break;

    default:
      printf ("No. %.2X\n",regs.r_ax);
    }
  }
else
  {
  if (regs.r_bx==3)
    printf ("Device");
  else
    printf ("Drive");

  printf (" %s connected to %s\n",devname,extname);
  }
}
```

APPENDIX 6 - REDIR SOURCE CODE.

```
;       Catch network redirection.
;AMENDMENTS:
;v1.0           Early Feb. '92
;       Quickly knocked up to see whether it works or not.

;v1.1           20/2/92
;       Pointer to data buffer included behind INT 21h vector so that REDIRSHW
;       can read it, rather than having to use SYMDEB. Message added.

;v1.2           11/3/92
;       General cleanup, i.e. exit via INT 21h function 31h, improved buffer
;       control, etc.

cseg    segment para public 'code'

        assume  cs:cseg, ds:cseg, es:cseg, ss:nothing


                mov     ax,3521h
                int     21h             ; get int 21 vector
                push    cs
                pop     ds
                mov     dx,offset Message
                mov     ah,9        ; print message
                int     21h
                mov     word ptr oldvectseg,es
                mov     word ptr oldvect,bx         ; store at JMPF instruction
                mov     dx,offset newhandler
                mov     ax,2521h     ; set int 21 vector
                cli
                int     21h
                sti
                mov     dx,offset LASTBYTE
                shr     dx,1         ;convert to paragraphs
                shr     dx,1
                shr     dx,1
                shr     dx,1
                add     dx,1         ;don't lose fraction
                mov     ax,3100h
                int     21h      ; tsr

Message:        db 'REDIR.EXE v1.2 (C) 1992 David Woodhouse',13,10
                db 'Catches network redirections.',13,10
                db 'Now installed. Use REDIRSHW to list redirections.',13,10,'$'

                dw      buffer       ; data for REDIRSHW
bufptr:         dw      buffer
idmsg:          db      '(C)DW',0
newhandler:     pushf
                cmp     ax,5f03h
                jnz     goback            ; not a redirect command

                push    ax
                push    cx
                push    si
                push    ds
                push    di
                push    es

                push    cs
                pop     es
                mov     di,word ptr cs:bufptr   ; point to buffer

                cmp     di,offset buffer+0f80h
                ja      bufferfull
```

```
loop1:          lodsb                       ; store localname
                stosb
                or      al,al
                jnz     loop1

                pop     ds
                pop     si
                push    si
                push    ds

loop2:          lodsb
                stosb
                or      al,al
                jnz     loop2

loop3:          lodsb
                stosb
                or      al,al
                jnz     loop3           ; store netname+password

                mov     word ptr cs:bufptr,di

bufferfull:     pop     es
                pop     di
                pop     ds
                pop     si
                pop     cx
                pop     ax

goback:         popf

                db      0eah                    ; JMPF
oldvect:        dw      0
oldvectseg:     dw      0


buffer:         db      1000h dup (0)           ;Shouldn't actually do this -
LASTBYTE        db      0                       ;LASTBYTE equ $+1000h would be
                                                ;better, but I can't be
                                                ;bothered to check whether it
                                                ;would actually work.
                                                ; DW v1.2
cseg            ends
                end
```

APPENDIX 7 - REDIRSHW SOURCE CODE.

```
#include <string.h>
#include <dos.h>
#include <stdio.h>

int main ()
  {
  char far *int21;
  char far *bufend;
  unsigned seg,ofs;

  printf ("REDIRSHW.EXE (C) 1992 David Woodhouse\n"
        "Shows redirections caught by REDIR.EXE\n");

  int21=(char far *)getvect(0x21);

  if (strcmp(int21-6,(char far *)"(C)DW"))
    {
    printf ("Error: REDIR.EXE not first in INT 21 chain.\n");
    exit (1);
    }
  else
    {
    seg=FP_SEG(int21);
    ofs=*(unsigned far *)(int21-10);
    bufend=(char far *)MK_FP(seg,*(int far *)(int21-8));
    int21=MK_FP(seg,ofs);

    while (int21<bufend)
      {
      int21+=printf("%Fs ",int21);
      int21+=printf("%Fs ",int21);
      int21+=printf("%Fs\n",int21);
      }
    }
  }
```

APPENDIX 8 - MYSHELL SOURCE CODE.

```
#include <process.h>
#include <stdio.h>
#include <string.h>
#include <dos.h>

struct REGPACK regs;

void Connect (char *devname, char *netname);

void main()
  {
  int tries=0;

  printf ("\n\nNetwork entry for David Woodhouse. (C) 1992\n");
  printf ("Fixed 21/2/92.\n");
  while (tries++<3 &&
  stricmp(getpass("This grants access to my own directories. Type password:"),"WOODY"))
    {
    printf ("Sorry - Access Denied. Try again.\n\n");
    }
  if (tries==3) __emit__(0xea,0x00,0x00,0xff,0xff);    /* boot */

  Connect ("N:","\\\\SERVERA\\49133088""\0""43201990");
  Connect ("P:","\\\\SERVERA\\PUBLIC""\0""OBLONGAT");
  Connect ("Q:","\\\\SERVERA\\PUBX\0");
  Connect ("R:","\\\\SERVERA\\38609157""\0""34705118");
  Connect ("Z:","\\\\SERVERZ\\PUBLIC""\0""OBLONGAT");
  Connect ("LPT1","\\\\SERVERA\\LQ850\0");
  Connect ("LPT2","\\\\SERVERJ\\LQ850\0");
  Connect ("LPT3","\\\\SERVERZ\\PSTSCR\0");

  while (execl ("A:\\COMMAND.COM", "", "/p", NULL)==-1)
    {
    printf ("\007Error - A:\\COMMAND.COM not found. Insert disk and press a key.\n\n");
    getch();
    }

  }


void Connect (char *devname, char *netname)
  {
  regs.r_bx=(devname[1]==':')?4:3;     /* set printer/drive flag in BL */

  regs.r_cx=3;       /* arbitrary parameter - why not 3? */
  regs.r_ax=0x5f03;
  regs.r_ds=FP_SEG(devname);
  regs.r_si=(unsigned)devname;
  regs.r_es=FP_SEG(netname);
  regs.r_di=(unsigned)netname;

  intr (0x21, &regs);
  if (regs.r_flags&1)
    {
    /* carry set - error */
    printf (" Error connecting %s to %s\n",devname, netname);

    switch (regs.r_ax)
      {
      case 0x35:
        printf ("Network path not found.\n");
        break;

      case 0x56:
        printf ("Access denied.\n");
        break;
```

```
    case 0x55:
      if (devname[1]==':')
        printf ("Drive");
      else
        printf ("Device");
      printf (" in use.\n");
      break;

    default:
      printf ("No. %.2X",regs.r_ax);
    }
  }
else
  {
  if (regs.r_bx==3)
    printf ("Device");
  else
    printf ("Drive");

  printf (" %s connected to %s\n",devname,netname);
  }
}
```

APPENDIX 9 - WOMBATS SOURCE CODE.

```
;      TITLE        WOMBATS.SYS to grab SYSINIT module from top of mem
;      VERSION      1.0
;      DATE         20/3/92
;       (C) 1992 David Woodhouse

cseg       SEGMENT para public 'code'

           org 0

           assume cs:cseg,ds:cseg,es:nothing,ss:cseg

main:      dw 0ffffh         ;next dvc
           dw 0ffffh
           dw 8000h          ;attr.
           dw dvcstrat       ;device strategy
           dw dvcint         ;device interrupt
           db "GSysInit"     ;mfg name/unit

msg0       db 0Dh,0Ah,"WOMBATS.SYS",0Dh,0Ah,00


jmptable  equ $

           dw INIT           ;init
           dw noop           ;media check
           dw noop           ;build bpb
           dw noop           ;ioctl input
           dw noop           ;input
           dw noop           ;input, non-destruct
           dw noop           ;input status
           dw noop           ;input flush
           dw noop           ;output
           dw noop           ;output, verify
           dw noop           ;output status
           dw noop           ;output flush
           dw noop           ;ioctl output


rqheadr    dd 0              ;request header ptr

;page


                             ;device strategy
dvcstrat proc far

  mov cs:word ptr rqheadr,bx    ;save request header ptr
  mov cs:word ptr rqheadr+2,es
  ret

dvcstrat   endp


                             ;device interrupt
dvcint   proc far

  push ax                    ;save all regs
  push bx
  push cx
  push dx
  push di
  push si
  push bp
  push ds
  push es
```

```
  mov bx,cs
  mov ds,bx
  les di,cs:rqheadr          ;es:di --> req. header
  xor bx,bx
  mov es:[di+3],bx           ;status word = 0
  mov bl,es:[di+2]           ;get command
  cmp bl,13                  ;make sure in range
  cmc
  mov al,3                   ;assume unknown command error
  jc cont1                   ;if bad command -->
  shl bx,1                   ;bx = index to jmp address
  cld
  call word ptr cs:[jmptable+bx] ;do command


                             ;commands all return here

cont1:
  les di,cs:rqheadr          ;es:di --> req. header
  mov ah,2                   ;set done bit
  rcr ah,1                   ;rotate in cy flag for error bit
  or es:[di+3],ax            ;or in error condition
  pop es                     ;restore all regs
  pop ds
  pop bp
  pop si
  pop di
  pop dx
  pop cx
  pop bx
  pop ax
  ret

dvcint endp


;page

master proc near

;........................... INIT: ........................................

Init:
  push cs
  pop es
  mov ax,9000h
  mov ds,ax
  mov cx,1000h
  mov di,offset buffer
  mov si,0
  cld
  repz movsb              ;grab sysinit code
  mov si,sp

  mov ax,ss:[si+16h]
  call phword
  mov ax,0e3ah
  int 10h
  mov ax,ss:[si+14h]
  call phword
  mov ax,0e0dh
  int 10h
  mov ax,0e0ah
  int 10h
  mov ax,ss
  call phword
  mov ax,0e3ah
  int 10h
  mov ax,sp
  call phword
```

```
  les di,cs:rqheadr          ;point to request header
  mov word ptr es:[di+14],offset LASTBYTE  ;set last byte needed
  mov es:[di+16],cs


i2:
  ret

;page
;.......................... No Operation: ..................................

noop:
  mov ax,3                   ;unknown command
  stc
  ret




;......................phword.........................

;      IN:    AX     word to be printed
;      OUT:   nothing
;      LOST:  nothing

phword:             push   ax
              xchg   al,ah
              call   phbyte
              mov    al,ah
              call   phbyte
              pop    ax
              ret

;................phbyte...........

;      IN:    AL     byte to printed
;      OUT:   nothing
;      LOST:  nothing

phbyte:             push   ax
              mov    ah,al
              shr    al,1
              shr    al,1
              shr    al,1
              shr    al,1
              call   phnibble
              mov    al,ah
              call   phnibble
              pop    ax
              ret

;..............phnibble................

;      IN:    AL     nibble to be printed
;      OUT:   nothing
;      LOST:  nothing

phnibble:     push   ax
              and    al,0Fh
              add    al,'0'
              cmp    al,'9'
              jna    ph1
              add    al,'A'-'9'-1
ph1:          mov    ah,0Eh
              int    10h
              pop    ax
```

```
            ret

;........................... Get setup parameters ............................

db '     BUFFER STARTS HERE:'
buffer equ $
LASTBYTE equ $+1000h
master     endp

cseg       ends
           end main
```

APPENDIX 10 - SHOWPARM SOURCE CODE.

```
;      TITLE        SHOWPARM.SYS to grab SYSINIT module from top of mem
;      VERSION      1.0
;      DATE         20/3/92
;       (C) 1992 David Woodhouse

cseg       SEGMENT para public 'code'

           org 0

           assume cs:cseg,ds:cseg,es:nothing,ss:cseg

main:      dw 0ffffh         ;next dvc
           dw 0ffffh
           dw 8000h          ;attr.
           dw dvcstrat       ;device strategy
           dw dvcint         ;device interrupt
           db "GSysInit"     ;mfg name/unit

msg0       db 0Dh,0Ah,"SHOWPARM.SYS",0Dh,0Ah,24h


rqhdr    dd 0              ;request header ptr


                             ;device strategy
dvcstrat proc far

  mov cs:word ptr rqhdr,bx    ;save request header ptr
  mov cs:word ptr rqhdr+2,es
  ret

dvcstrat   endp


                             ;device interrupt
dvcint   proc far

  push ax                    ;save all regs
  push bx
  push cx
  push dx
  push di
  push si
  push bp
  push ds
  push es

  les di,cs:rqhdr          ;es:di --> req. header
  mov es:[di+3],0100h           ;status word = done
  cmp byte ptr es:[di+2],0            ;is it INIT command?
  jnz cont1                  ;no -->
  call init
                             ;commands all return here

cont1:
  pop es                     ;restore all regs
  pop ds
  pop bp
  pop si
  pop di
  pop dx
  pop cx
  pop bx
  pop ax
  ret
```

```
dvcint endp


;........................... INIT: ......................................

Init:
       push   cs
       pop    ds
       mov    dx,offset msg0
       mov    ah,9
       int    21h                 ;print signon msg

       lds    si,es:[di+12h]             ;load pointer to command line
loop:  mov    ax,0e20h
       int    10h                 ;print space
       lodsb
       call   phbyte
       cmp    al,0ah
       jnz    loop
       mov    word ptr es:[di+0eh],0000   ;set last byte needed
       mov    es:[di+10h],cs
       ret

;......................phword..........................

;      IN:    AX     word to be printed
;      OUT:   nothing
;      LOST:  nothing

phword:             push   ax
              xchg   al,ah
              call   phbyte
              mov    al,ah
              call   phbyte
              pop    ax
              ret

;................phbyte...........

;      IN:    AL     byte to printed
;      OUT:   nothing
;      LOST:  nothing

phbyte:             push   ax
              mov    ah,al
              shr    al,1
              shr    al,1
              shr    al,1
              shr    al,1
              call   phnibble
              mov    al,ah
              call   phnibble
              pop    ax
              ret

;..............phnibble................

;      IN:    AL     nibble to be printed
;      OUT:   nothing
;      LOST:  nothing

phnibble:     push   ax
              and    al,0Fh
              add    al,'0'
              cmp    al,'9'
              jna    ph1
              add    al,'A'-'9'-1
ph1:          mov    ah,0Eh
              int    10h
```

```
              pop    ax
              ret

;........................... Get setup parameters ............................


cseg        ends
            end main
```