Hacking the IBM-PC

Course manual

“Digging into it”

Written by Vincent Himpe
Lab Design Center

Alcatel Microelectronics
ExcelsiorLaan 44-46
1930 Zaventem

Copyright notice.

Hacking the IBM-PC
“ Digging into it “

Original Release 06/09/1998

Design Lab Alcatel Microelectronics
Excelsiorlaan 44-46
B 1930-zaventem
Belgium

e-mail : vincent.himpe□mie.alcatel.be
intranet : file://tmp_mnt/disk222/ds/labman//index.html

Index table

Introduction.

Hello and welcome again. Apparently you have decided once again to broaden your horizon.

As a wise man once said :

> Any sufficiently advanced technology is indistinguishable from magic
> Arthur C. Clarke

Well the time has come to unravel the magic of the IBM-PC .The goal of this course is to provide a bit more insight in the way the IBM-PC works. Attention will go both to the real hardware of the machine , extending it and the software interface to it.

A number of tools will be presented that can help you in exploring the IBM PC. Most tools are already available on the machine itself.

This is not an education course to make you an Assembler-Wiz or Hardware Guru. The goal is to provide you with a bit more background information on the machine and to show you how you can easily find information you need.

Chapter 1 : The PC : a historical overview

The PC was first conceived as ‘a smart input terminal’ . It was never intended to be used as a standalone machine. When Don Estridge and his team of 13 started this project the goal was to make a small , smart terminal that could tun some front end software. The idea was to unload the big Mainframe computers from the task of serving consoles.

The project was to be an open-architecture low budget kind of thing. Most of the work was done by external companies. They started building the PC using a S100 bus computer board from Intel , a Monitor program previously written for a 6802 CPU from Motorola and a CP/M version for 8086 .

Starting from these parts and some experimenting a final schematic was drawn and the monitor program was extended to become , as we know it today , the BIOS. At that point it became clear that CP/M was not the way to go. IBM had seen a demonstration of an operating system developed by Seattle Computer products. Called QDOS. One of the design team members talked about this to the young Bill Gates. This guy joined the team as an external solution provider. It was agreed that Microsoft would port QDOS to the hardware platform of the PC and then modify an earlier written Basic interpreter (for Tandy corporation) to run on this platform.

When the PC was announced in 1981 PC DOS was it’s primary operating system. Outside market researchers pointed out that this project was doomed ! . Who would buy a computer that was not attached to a mainframe !.

Well it looks like they were wrong . Now , 17 years later the situation is the opposite : The mainframe has been moved to the museum a long time ago .Every office holds more computing power then the average mainframe of 1981.

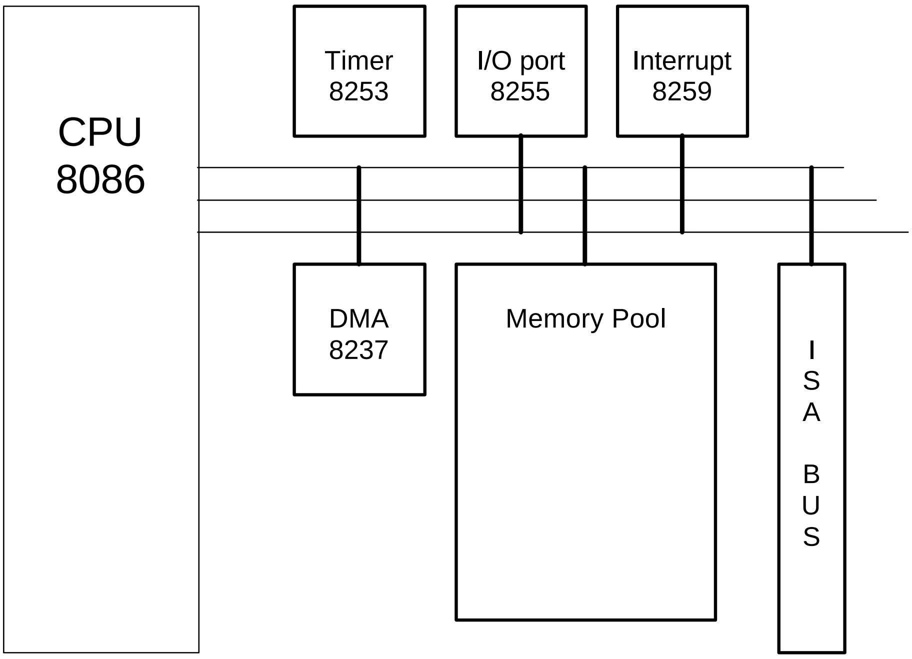
Besides the software the hardware has improved substantially. Where the first machine ran PC DOS 1.0 , today’s machines run Windows / Windows NT / UNIX and clones and every other possible operating systems. Literally millions of applications have been developed , both commercial and shareware / freeware.

This machine has set off what has been called ‘the computer revolution’

The goal of this course is to provide some insight on how this whole machine works both from a hardware and a software point of view.

Chapter 2 : The PC : a hardware description

The original IBM-PC hardware merely consisted of some standard chips that Intel was selling at that time . The block schematic showed the following components :



Original IBM PC topology

Besides the CPU , memory and some chips to get the thing running there was nothing else in the machine. Every PC , whether it is an original IBM-5100 from 1979 or the latest state of the art souped-up Dual Pentium-II 733 MHz, still adheres to this topology.

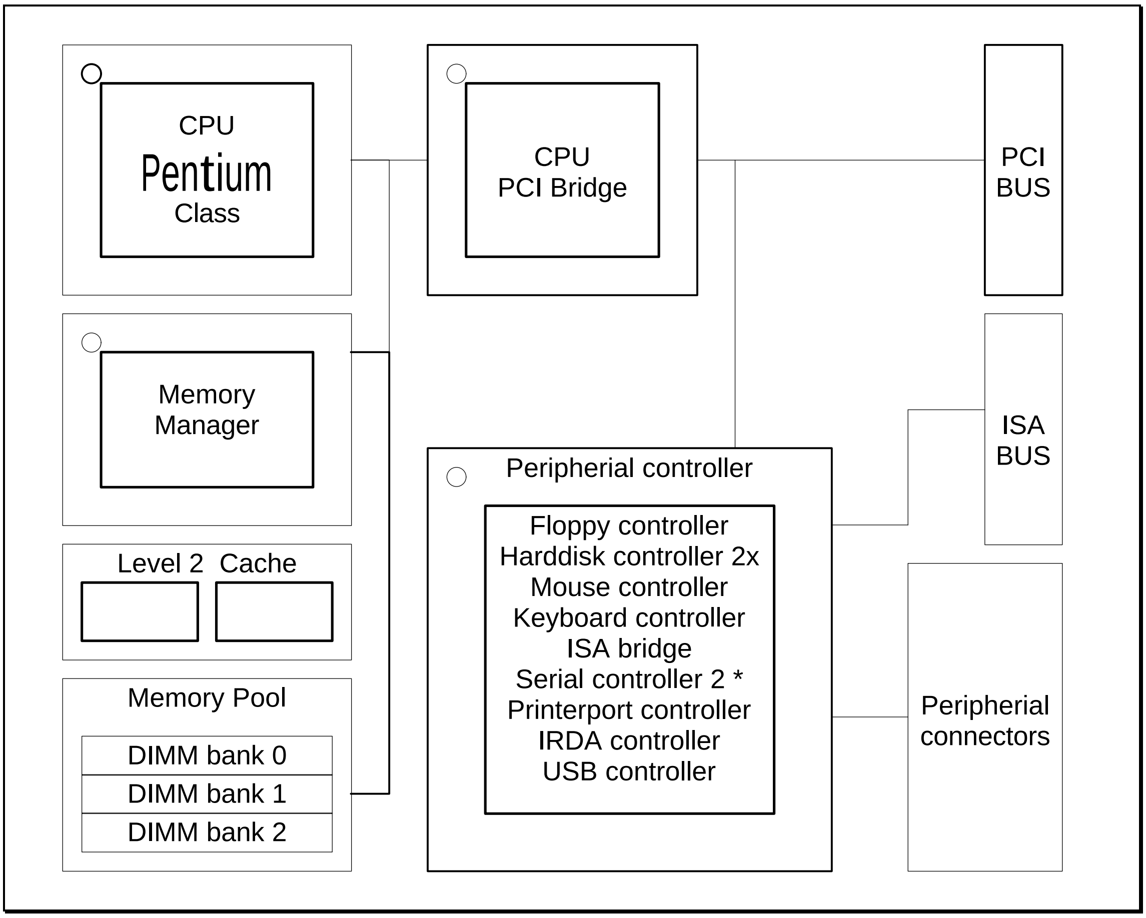
You will still find a 8253 Triple timer , one or two 8259 interrupt controllers and a DMA controller of the 8237 type inside your computer. Maybe they are no longer visible as such but they are in there somewhere. The only things that have really changed in the PC are the speed ,and the width of the data and address bus.

The speed of all components and ,markedly the speed of the CPU , has gone up tremendously. Where the original machine ran at a blazing 4.77 MHz , today's machines run at 400 MHz . that is a 100 fold performance !.

This had lead to the fact that new techniques had to be developed to cope with the new speed-demons.

Things such as second level and third level cache have been designed. , new buses emerged (Vesa Local bus , PCI) , and new communication standards have been set forth (IRDA , PCI ,Firewire , IEEE1384).

But apart from this , the PC looks still the same.
A modern PC block diagram looks somewhat like this. The astonishing fact is that you can translate this directly to a component schematic:



The entire PC has been scaled down to a mere 4 IC's. You still have the CPU which is now of the Pentium - Class . (Pentium / Pentium Pro / Pentium II all with or without MMX) . Besides this you need 3 more components to build a computer : The memory controller. This component is a single chip that handles all accesses from the CPU to the memory . It takes care of refresh cycles , cache update , and so on.

A second chip builds an interface between the CPU and the PCI local bus. The signals coming from the PCI bridge are fed to the backplane connectors. For your convenience most manufacturers also put a so called 'multi-io' chip on the main board. This one contains all the peripherals that , in the original PC , were previously on Plug-in boards.

Just add memory and you are ready to run.

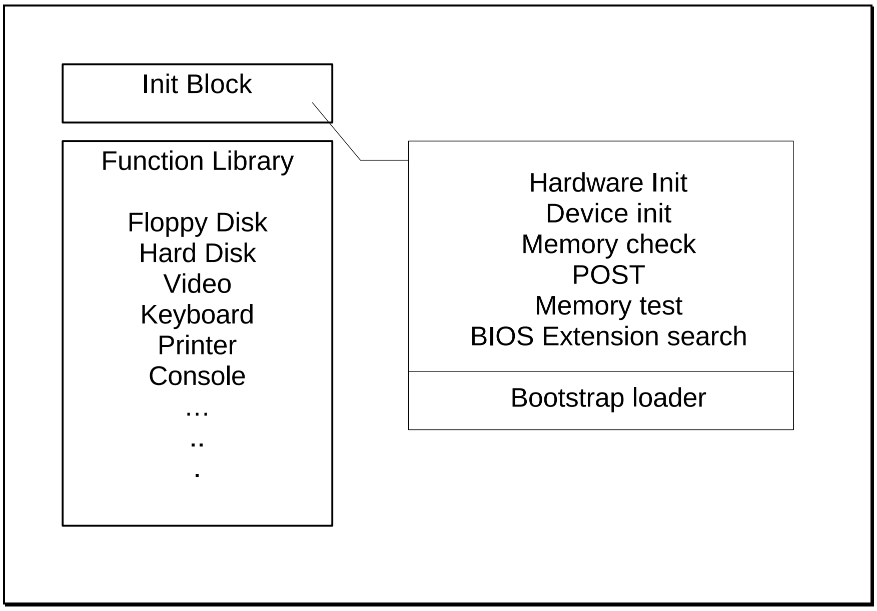
Chapter 3 : The BIOS

Ahh .The BIOS. The Basic Input / Output System. Without this your computer would do absolutely nothing. You would only get a black screen at startup , no flashing cursor , no beep , no nothing. This is the little thingie that gets you machine up and running , and allows it to load an operating system of your choice.

The BIOS resides in ROM on the main board. Nowadays modern boards use so called Flash-ROM . This means that whenever an update comes for the BIOS you can overwrite the old one. After all the BIOS is also a piece of software that might have bugs and is subject to revisions.

The BIOS generally contains 3 parts :

- A system initialization and Test routine called the POST routine.
- A huge library of function to control hardware devices such as FDD / HDD / COM / SERIAL /VIDEO adapters.
- A Bootstrap Loader.



The first task of the BIOS is to initialize the hardware on the main board to an operating state. After Power-Up and Reset the CPU Accesses memory location F000:0000 and starts executing whatever it finds there. And that is .. exactly the BIOS .

The BIOS program initializes the register of the standard PC components to an operational state (DRAM refresh , timers , DMA and interrupt handlers. After that the BIOS performs an memory boundary check. It simply tests how much memory there is in the 0.. 640 k range. On modern machines if it finds sufficient memory it will make a working copy of itself into RAM memory . This improves the speed drastically.

The next steps depend now on the BIOS manufacturer. In general some sort of complete memory test is done , the hardware detected etc .. . Modern Plug and Play BIOS systems that handle PCI devices can arbitrate those boards too.

When all is done the BIOS will execute the Bootstrap loader. This loader simple checks all disks detected by the POST for a signature on sector 0 . If the correct signature is found the entire Track is read into memory starting at address 0 and then execution is handed over to that address. This is called ‘loading the Boot sector’

The real work of the BIOS is now over. At this point the Boot loader loaded from takes over and will start loading an operating system. The boot loader of any operating system , whether it is DOS / Windows / Windows NT / Linux / Solaris etc . is stored in this location. This bootloader is simply a binary image of a program . More about the boot loader in the next chapter.

The only part we have not discussed yet is the library part. This is a collection of routines that allows software to access the hardware. After all , there are thousands of different motherboards out there with a lot of different components. Well the task of the BIOS is to provide an interface between the software and this hardware. Standard tasks such as sending a character to the display are handled by the BIOS .

These functions are used by DOS programs . More modern operating systems such as Windows etc .. access the hardware directly.

Now there is one more trick up the sleeve. The BIOS is extendible !. Let’s take a look at the VIDEO routines. There are thousands of different boards out there. No single BIOS can contain all function that support all possible boards. Therefore the BIOS was designed to be extendible. To do this the POST scans for signatures in memory banks D and E (D000:0000 .. EFFF:FFFF).

The 0x55h 0xAAh signature has to be on a 2K boundary inside this range. Whenever this signature is found then the execution is handed over to this code.

```
-d c000:0
C000:0000  55 AA 40 EB 04 37 34 30-30 E9 E6 1F 00 00 00 00   U.@..7400.......
C000:0010  00 00 00 00 00 00 00 00-38 01 00 00 00 00 49 42   ........8.....IB
C000:0020  4D 20 56 47 41 20 43 6F-6D 70 61 74 69 62 6C 65   M VGA Compatible
C000:0030  20 42 49 4F 53 2E 20 BB-66 19 01 8A 01 D0 00 00    BIOS. .f.......
C000:0040  FF 00 03 20 20 53 74 65-61 6C 74 68 20 36 34 20   ...  Stealth 64
C000:0050  56 69 64 65 6F 20 56 52-41 4D 20 56 65 72 73 2E   Video VRAM Vers.
C000:0060  20 33 2E 30 30 20 20 28-63 29 20 44 69 61 6D 6F    3.00  (c) Diamo
C000:0070  6E 64 20 4D 75 6C 74 69-4D 65 64 69 61 20 53 79   nd MultiMedia Sy
-
```

As you can see in the above image the Video BIOS typically resides at C000:000

Other BIOS extensions can be SCSI or any other custom device.

The main BIOS resides at F000:0000

```
-D f000:0000
F000:0000  EB 01 90 33 DB 8E DB C7-87 72 04 35 12 E9 33 B9   ...3.....r.5..3.
F000:0010  43 4F 50 59 52 49 47 48-54 20 28 63 29 20 31 39   COPYRIGHT (c) 19
F000:0020  38 39 2D 39 36 2C 20 4D-49 43 52 4F 49 44 20 52   89-96, MICROID R
F000:0030  45 53 45 41 52 43 48 20-49 4E 43 2E 20 2D 20 4D   ESEARCH INC. - M
F000:0040  52 20 42 49 4F 53 20 28-72 29 20 2D 20 41 4C 4C   R BIOS (r) - ALL
F000:0050  20 52 49 47 48 54 53 20-52 45 53 45 52 56 45 44    RIGHTS RESERVED
F000:0060  12 4D 52 20 42 49 4F 53-20 28 72 29 20 20 56 33   .MR BIOS (r)  V3
F000:0070  2E 33 30 1A 43 6F 70 79-72 69 67 68 74 20 28 63   .30.Copyright (c
-
```

The above image shows the start of a Microid Research BIOS as found on Intel Endeavor Boards. Remind you that this is the real BIOS . So you will not see the signature 0x55 0xAA here

What you can see of the previous examples is that all of these BIOS programs contain an identification string.

These BIOS extensions also consist of 2 parts. You have to initialization program and the library block. The initialization block might change interrupt vectors to allow the routing of standard functions towards the new libraries. The library contains the board and vendor specific routines.

For example a video board. Nowadays all video boards adhere to the VGA standard.
But the original BIOS can only send characters to a text (MDA / Hercules) card. The VGA board does not understand the bytes coming in from the main BIOS. Therefore the Video BIOS has the correct functions that allow you to send a character to the screen.

Upon boot the addresses of these routines will be replaced by the addresses of the routines in the extended BIOS

How can you find these BIOS extensions ? Well simple using DEBUG of course.
For more information on debug consult the section about this superb program

Chapter 4 : The BIOS system Area

When the BIOS has completed it’s task and handed over control to the bootstrap loader it left some data in memory. This data is sometimes called the System area or the ‘SystemMetrics’ Data. It is a block of 256 bytes that gives information about the machine.

A typical page looks like this

```
-d 0040:0000 ff
0040:0000  F8 03 F8 02 E8 03 E8 02-78 03 78 02 00 00 00 00   ........x.x.....
0040:0010  23 C8 00 80 02 00 30 A0-00 00 36 00 36 00 30 52   #.....0...6.6.0R
0040:0020  30 52 20 39 32 50 35 4C-35 4C 08 0E 08 0E 08 0E   0R 92P5L5L......
0040:0030  66 21 66 21 0D 1C 30 52-3A 34 30 52 30 52 01 00   f!f!..0R:40R0R..
0040:0040  04 00 20 00 00 00 00 00-00 03 50 00 40 20 00 00   .. .......P.@ ..
0040:0050  00 17 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0040:0060  07 04 00 D4 03 29 30 76-07 87 1C FF 84 6F 13 00   .....)0v.....o..
0040:0070  00 00 00 00 00 00 02 08 00-14 14 14 3C 01 01 01 01   ...........<....
0040:0080  1E 00 3E 00 31 08 00 60-09 11 0B 80 58 00 00 07   ..>.1..`....X...
0040:0090  87 07 00 00 00 00 10 12-A0 00 40 00 88 FD FF FF   ..........@.....
0040:00A0  00 00 00 00 00 00 00 00-2E 39 00 C0 00 00 00 00   .........9......
0040:00B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0040:00C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0040:00D0  00 00 00 00 00 00 00 00-00 00 00 01 00 00 00 30   ...............0
0040:00E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0040:00F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

Most of this stuff is irrelevant for what we do. However some very interesting things can be found.

Let’s take a look at the very first line

```
0040:0000  F8 03 F8 02 E8 03 E8 02-78 03 78 02 00 00 00 00   ........x.x.....
```

This line gives us information about the available ports in the machine and about the address they are locked on. The first four words specify the serial ports and the next 3 words specify the parallel ports. The last word is undetermined. Some machines use it for a fourth printerport but in general it is not in use.

Note :

A port address is specified by 2 bytes in little endian coding. This means that , in order to obtain the correct address , you have to swap the high and low byte. Example F8 03 becomes 0x03f8

From left to right we find this 0x03f8 0x02f8 0x03e8 0x 02e8 for the serial ports and 0x0378 and 0x0278 for the parallel ports. This means on this particular machine there are 4 serial ports and 2 parallel ports. If a port locator contains 0x0000 it means that this port does not exist. The BIOS functions (and most DOS programs and) use these tables to look up the address for their transactions. This means that , if you want to put a LPT port at a certain address , you can plug in the board and specify the address here yourself. Any printer routine will then be redirected to this new address.

Another thing you can find here is the keyboard buffer . This buffer starts at location 0040:001e and is 32 bytes long. You can even see the status of Num. lock and caps lock LED’s on your keyboard.

From the above you can see that this area is not simply a ‘dead’ zone. Whenever the BIOS executes an operation it can return information for general use in this area. As this contains so much information it is impossible here to give an explanation of all of the bytes , words and bits stored therein. If you need to know more please consult a book that details on this (see end of course for a suggested reading list.

Neat tricks you can perform with the BIOS area

A little example : You have 2 printers each on a printerport. But you have a program that can only print to port 1 . Now you want to print on the printer attached to port 2. Are you going to swap cables ? . No you simply fire up debug and modify the address selector of the first printerport to that of the second printerport. Exit debug and start your program. Presto.

```
-d 0040:000
0040:0000  F8 03 F8 02 E8 03 E8 02-78 03 78 02 00 00 00 00   ........x.x.....
0040:0010  23 C8 00 80 02 00 30 A0-00 00 34 00 34 00 64 20   #.....0...4.4.d
0040:0020  20 39 30 52 30 52 34 4B-30 52 3A 34 30 52 30 52    90R0R4K0R:40R0R
0040:0030  30 52 0D 1C 30 52 30 52-0D 1C 3F 32 0D 1C 01 00   0R..0R0R..?2....
0040:0040  6F 00 20 00 00 00 00 00-00 03 50 00 40 20 00 00   o. .......P.@ ..
0040:0050  00 31 00 00 00 00 00 00-00 00 00 00 00 00 00 00   .1..............
0040:0060  07 04 00 D4 03 29 30 76-07 87 1C FF 19 97 13 00   .....)0v........
0040:0070  00 00 00 00 00 02 08 00-14 14 14 3C 01 01 01 01   ...........<....
-e 0040:0008 78 02 78 02
-d 0040:00
0040:0000  F8 03 F8 02 E8 03 E8 02-78 02 78 03 00 00 00 00   ........x.x.....
0040:0010  23 C8 00 80 02 00 30 A0-00 00 2E 00 2E 00 30 52   #.....0.......0R
0040:0020  30 52 34 4B 30 52 3A 34-30 52 30 52 0D 1C 30 52   0R4K0R:40R0R..0R
0040:0030  30 52 3A 34 30 52 30 52-0D 1C 64 20 20 39 01 00   0R:40R0R..d  9..
0040:0040  B5 00 20 00 00 00 00 00-00 03 50 00 40 20 00 00   .. .......P.@ ..
0040:0050  00 31 00 00 00 00 00 00-00 00 00 00 00 00 00 00   .1..............
0040:0060  07 04 00 D4 03 29 30 76-07 87 1C FF D3 98 13 00   .....)0v........
0040:0070  00 00 00 00 00 02 08 00-14 14 14 3C 01 01 01 01   ...........<....
-
```

As you can see the LPT1 was originally set to 0x0378 and lpt2 was set to 0x0278 . After our little operation they have changed places.

Note that this trick does not work for Windows programs. Windows relies on it’s own SystemMetrics page which is completely different and inaccessible. However this trick still does work in a DOS box running on Windows.

Chapter 5 : The I/O components

The previous chapter has shown you how you can find the address and number of ports in your machine. Now let's talk a bit more about the ports themselves.

5.1 The Parallel port

This is the standard parallel port that you use to connect a printer. This port is often referred to as a Centronics interface or Printerport. The PC has the capability to detect 3 parallel ports

A parallel port consists of 3 subsequent registers. The Output register is located at the address you can find in the SystemMetrics memory area .

Base	D7	D6	D5	D4	D3	D2	D1	D0	Output Register
	9	8	7	6	5	4	3	2	
Base + 1	BS	AQ	OP	SL	ER	-	-	-	Status Register
	11	10	12	13	15	-	-	-	
Base + 2	-	-	-	IE	SI	IP	AF	ST	Control Register
	-	-	-	-	17	16	14	1	

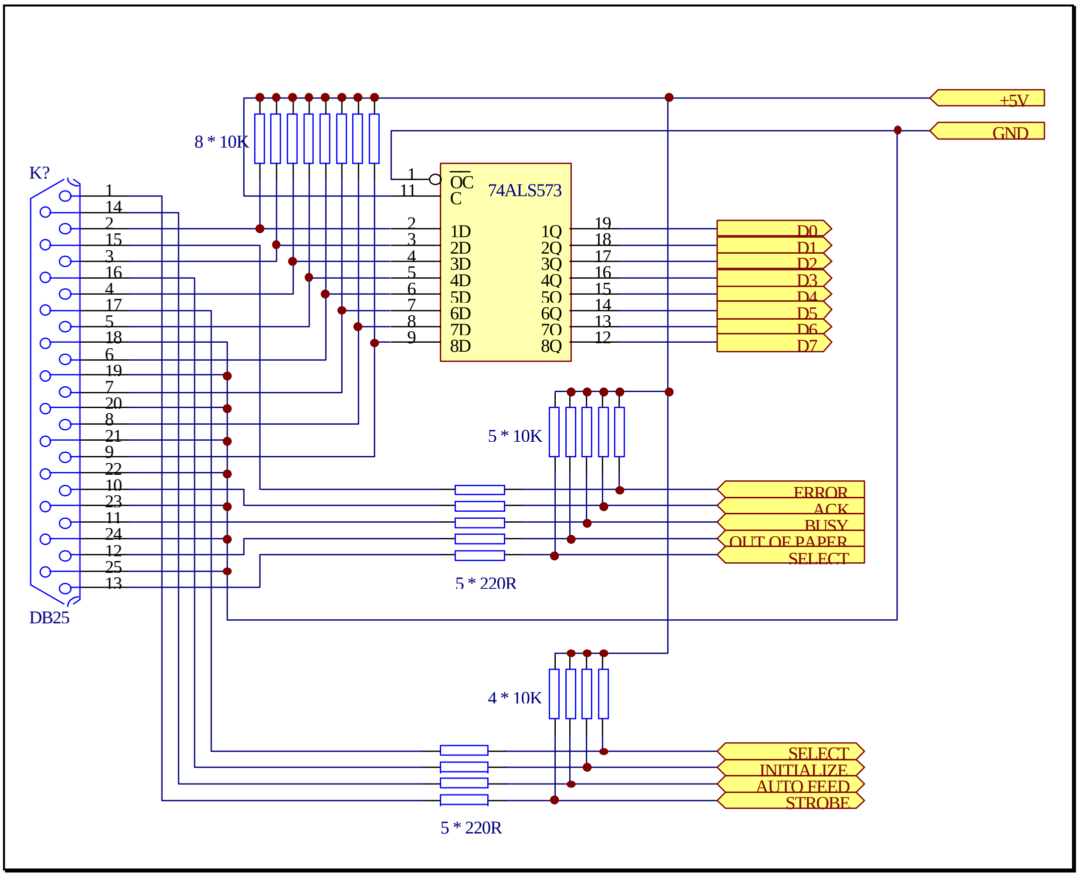
The above table shows the register map for a standard Printerport. These ports can per default be found in any PC. A typical PC has at least one port (mostly on 0x378 , sometimes on 0x3bc)

Definition of symbols used.

BS	Busy	Input
AQ	Acknowledge	Input
OP	Out of Paper	Input
SL	Select	Input
ER	Error	Input
IE	IRQ enable	Input
SI	Select input	Bidir
IP	Initialize Printer	Bidir
AF	Auto Feed	Bidir
ST	Strobe	Bidir

The numbers under every bit correspond to the pin number on the connector. Modern printerports have features that allow it to behave fully bi-directional. However , since not all machines use these ports it is not recommended to use these advanced modes. Furthermore the control register can be 'floating'. This means that it is undetermined where the register gets mapped that controls this bi-directional port. It depends on the driver loaded by the BIOS and or operating system.

As you can see from the above , the printerport opens up a lot of possibilities to control boards. However , since a printerport is part of the computer , care should be taken not to destroy it accidentally . The following schematic shows a typical interface which will protect the port under most circumstances. It also acts as a buffer to clean up the signals generated by the port



Printerports can be found on 3 addresses in the machine. Normally the first printerport resides at 0x378 . You can find the correct addresses and the amount of ports in you computer by snooping around in the BIOS data area at segment 0040. Check out the chapter about the use of Debug to find out more.

5.2 The Serial Port

The Serial port is the second port which is standard port available on any computer. There are two different styles. You can have either the full fledged 25 pin connector or the shrinked 9 pin connector. The port is controlled by a UART of the 8250 / 8251 or 165x0 type. The latter has more advanced features like transmit and receive FIFO's. however from a programmers point of view these controllers all look the same.

Since the port holds a lot of registers and is quite complicate to control low level , I'm not going to detail on that here and now. I will explain the things you need to know and how you can make an interface that works reliable all the time.

While the UART can be set to all sorts of different baudrates , parities , stopbits , modes and so on , the only one that is really important is 9600,n,8,1 mode. This is a typical communication mode that is most widely used to talk to devices of all sorts.

Now what exactly does it mean. Well simple :

9600	Baudrate	measured in Bits per second
N	Parity	No parity. Alternative settings could be O(dd) or E(ven)
8	Databits	The number of bits transmitted at a time . Can be set anywhere between 5 and 11
1	Stopbits	Number of bits added to close the packet can be set to 1 2 or ½

You can set a port to these parameters by using the mode command from DOS.. Check out the section about the Mode command to learn more.

Port interface

A typical serial interface needs only 3 wires . TX , RX and Ground. To prevent data loss the RS232 also has a set of handshake lines. These lines can be used to control the interaction between devices. However , in most applications where you connect a device to a Serial port these lines are not used. However ,some programs check these lines and simply do nothing. But fortunately you can trick them .

Null modem cable

To connect two device together you will need what is called a Null Modem cable. This is the most simple cable to connect 2 devices.

2	-	3
3	-	2
5	-	5
1 4 6		1 4 6
7 8		7 8

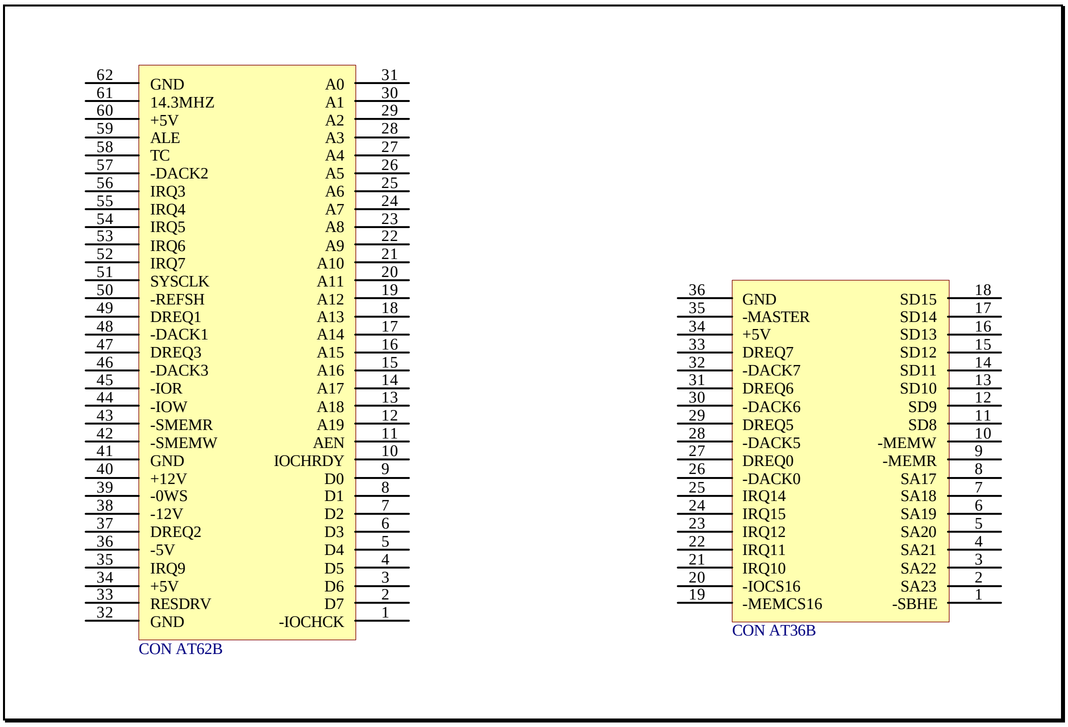
If make a cable with the above pinning (9 pole D connector) , then you can connect virtually any device to a serial port.

5.3 The ISA bus

This is the core of the IBM PC. It was selected as the adapter connection bus for the IBM Datamaster 5100. When IBM designed the PC 2 years later they kept the same bus and pinning. Actually the interface board of the Datamaster would work in the PC and vice versa. Today this bus is loosing popularity to the PCI bus. The reason is the limited throughput .

The bus clock frequency is 14.737 MHz. Because some cycles are needed to perform a complete transaction the real bus speed lies somewhere around 3 to 4 MHz. In the old machines you could even fit memory into the IO range. This would steal a lot of IO cycles and decrease the real bus speed a lot. (such boards were call Above Boards or RamDisk)

When the 286 machines were introduced an extension was made to the buses to allow access to the additional data and address lines available in 16 bit machines.



The picture above shows both connectors and their pinning. Describing on how to make a full interface would go way too far for this course.

5.4 The Vesa Bus

This bus was designed by the VESA organization (Video Electronics Standardization Association) . The emerging of new GUI based systems like Windows demanded a dramatical improvement in bandwidth . Especially the Graphics (read : video) access need a boost. This organization set forth a bus specification to cope with these problems. Actually the VESA bus is not a real bus. It is merely a direct access port to the processors signals. By providing this access the slow ISA bus could be bypassed.

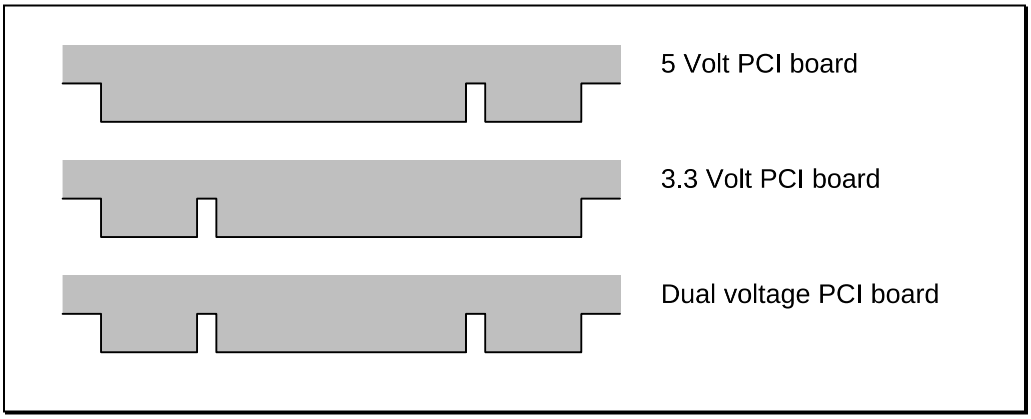
Besides Video boards and hard disk controllers nobody really made VESA boards. The recent rise of the PCI bus has made the VESA bus obsolete.

5.5 The PCI bus

This is the new emerging standard on the Local Bus frontier. While originally designed by , yes Intel again , this bus has gained global industry acceptance. Even machines from manufacturers like Apple , Sun , Hp ,Silicon graphics etc have adopted this bus as a standard. This means that you can exchange the plug - in board between all of these hardware platforms (provided you find the drivers and software for them)

The PCI bus is actually a collection of Buses. Furthermore PCI is a standard that is evolving. The current release 2.0 has a 64 bit capability where the original standard only defined a 32 bit bus. The physical connector has remained the same. Due to the high speed nature of the bus (up to 132 Mbytes/s) the electrical characteristics are carved in stone. This makes it hard to design an interface board on PCI. You will need a special controller chip that allows you to interface to the BUS backplane

Because of the complexity of hardware design for this bus an in depth discussion will not be given.



Chapter 8 : Setting OS parameters : MODE

The MODE command allows you set parameters that alter the way you r machine regards to it’s hardware. This command talks directly to the system BIOS and / or loaded extensions.

There are a number of parameters that allow you to control the behavior of the computers communication ports. The behavior of the console can be altered if the ANSI.SYS driver is loaded into memory.
Firing up MODE will display the list of current settings.

```
Status for device LPT1:
-----------------------
LPT1: not rerouted
Retry = NONE
Code page operation not supported on this device

Status for device LPT2:
-----------------------
LPT2: not rerouted
Retry = NONE
Code page operation not supported on this device

Status for device LPT3:
-----------------------
LPT3: not rerouted

Status for device CON:
----------------------
Columns=80
Lines=50
Active code page for device CON is 850
Prepared code pages:
  code page 850

Status for device COM1:
-----------------------
Retry = NONE

Status for device COM2:
-----------------------
Retry =NONE

Status for device COM3:
-----------------------
Retry = NONE

Status for device COM4:
-----------------------
Retry = NONE
```

The mode command allows you to alter the operational state of all communication channels. This means not only Printer and Serial ports but also some parameters of the Keyboard and Screen

If you launch the command MODE /? You will get a detailed list.

```
C:\WINDOWS>mode /?
Configures system devices.

Printer port:      MODE LPTn[:] [COLS=c] [LINES=l] [RETRY=r]
Serial port:       MODE COMm[:] [BAUD=b] [PARITY=p] [DATA=d] [STOP=s] [RETRY=r]
Device Status:     MODE [device] [/STATUS]
Redirect printing: MODE LPTn[:]=COMm[:]
Prepare code page: MODE device CP PREPARE=((yyy[...]) [drive:][path]filename)
Select code page:  MODE device CP SELECT=yyy
Refresh code page: MODE device CP REFRESH
Code page status:  MODE device CP [/STATUS]
Display mode:      MODE [display-adapter][,n]
                   MODE CON[:] [COLS=c] [LINES=n]
Typematic rate:    MODE CON[:] [RATE=r DELAY=d]
```

What we are interested in are the LPT and COM ports , possible the console and maybe the redirecting options it provides.

Lets start with the printerports.

The MODE command allows you to modify the way the BIOS printer routine (int 5) works. You can specify the width of the printer device (80 or 132 columns) , the spacing between lines and the Retry bit. The only interesting option is the Retry bit. If the printer is not responding in time then the BIOS will retry. By default this setting is OFF. This means : no printer -> data lost.

The Console.

The only interesting option here is to specify the Lines option. If you set this to 50 then you have double the number of lines then normal. Great for debugging sessions .

```
mode con lines=50
mode con lines=25
```

Serial ports:

The MODE command allows you to set the operational parameters of the UART . You can specify baudrate , parity ,the number of data and stop bits and the retry option.

```
Mode com1:9600,n,8,1
```

The above command sets the port to 9600 baud , no parity , 8 databits and one stopbit. In general the normal baudrates between 75 and 19200 are available on any machine. The higher baudrates are only possible on more advanced uarts like 16550 . However , DOS only supports up to 19200 . Why ? because the mode command only accepts integers . (Max 32768). Then next higher baudrate is 38400 baud. But there is also another reason. If you want to do such high speed communications you will need a completely different style of access to the ports.

Chapter 9 : Examining the memory : MEM

The MEM command allows you to retrieve information about the memory usage of your computer.

The bare usage of the MEM command only display the free amount of memory. However there are 2 very important modifiers you can use with the MEM command.

Parameter	Action
/D	Debug information . This displays status of all modules in memory, internal drivers, and other information.
/M	Displays a detailed listing of a module's memory use. This option must be followed by the name of a module, optionally separated from /M by a colon.

A typical output of the MEM command is shown below. However the real power of the MEM command lies in the modifiers that you specify.

```
C:\>mem

Memory Type        Total       Used       Free
----------------  --------   --------   --------
Conventional          640K        68K       572K
Upper                   0K         0K         0K
Reserved              384K       384K         0K
Extended (XMS)     31,744K       184K    31,560K
----------------  --------   --------   --------
Total memory       32,768K       636K    32,132K

Total under 1 MB      640K        68K       572K

Total Expanded (EMS)                   32M (33,030,144 bytes)
Free Expanded (EMS)                    16M (16,777,216 bytes)

Largest executable program size       572K (586,192 bytes)
Largest free upper memory block         0K       (0 bytes)
MS-DOS is resident in the high memory area.

C:\>_
```

The /D modifier shows a complete map of the things that are loaded into memory. The next figure shows part of this output . A typical output consists of multiple pages. Therefore it is a good idea to specify also the /P modifier to stop after a page. You can then later press a key to continue.

Tip :

You can also store the output of this, and many other DOS commands by redirecting it to a file. Example : MEM /M >tempfile . This will create a file called ‘tempfile’ and store the screen output in this file. You can then later retrieve it using EDIT or your favorite editor.

```
Conventional Memory Detail:

Segment               Total        Name         Type
-------          ----------------  -----------  --------
00000              1,024    (1K)               Interrupt Vector
00040                256    (0K)               ROM Communication Area
00050                512    (1K)               DOS Communication Area
00070              1,424    (1K)  IO           System Data
                                  CON          System Device Driver
                                  AUX          System Device Driver
                                  PRN          System Device Driver
                                  CLOCK$       System Device Driver
                                  A: - D:      System Device Driver
                                  COM1         System Device Driver
                                  LPT1         System Device Driver
                                  COM2         System Device Driver
000C9              5,184    (5K)  MSDOS        System Data
0020D             35,168   (34K)  IO           System Data
                   1,024    (1K)               Relocated EBIOS data
                   4,304    (4K)  CON          Installed Device=ANSI
00AA3                 80    (0K)  MSDOS        System Program
00AB9              4,688    (5K)  DOSKEY       Program
00BDE              6,944    (7K)  KEYB         Program
00D90                256    (0K)  WIN          Environment
00DA0              3,408    (3K)  WIN          Program
00E75              2,224    (2K)  vmm32        Program
00F00                272    (0K)  COMMAND      Data
00F11              5,728    (6K)  COMMAND      Program
01077              1,440    (1K)  COMMAND      Environment
026F9            495,728  (484K)  MSDOS        -- Free --
```

As you can see this gives a detailed overview of the live memory. Let’s llok a bit more in detail :

```
00AB9              4,688    (5K)  DOSKEY       Program
```

This kind of table is to be read from right to left . : A program called ‘DOSKEY’ is taking 5K of memory . the real physical size is 4688 bytes. It commences in segment 00AB9

Lets examine another part :

```
00040                256    (0K)               ROM Communication Area
```

The ROM communication area (resulting from the POST) contains 256 bytes and is located at segment 00040. This means you can now dig into the memory and examine the contents of it. Therefore we need a more advanced tool called DEBUG. More about DEBUG in the next chapter.

First let’s see what the /M modifier does.

```
C:\>mem /M COMMAND

COMMAND is using the following memory:

 Segment  Region       Total        Type
 -------  ------  ----------------  --------
  01800              288    (0K)  Data
  01812            8.624    (8K)  Program
  01A2D            1.440    (1K)  Environment
                  ----------------
 Total Size:        10.352   (10K)
```

As you can see this command allows you to examine one particular program and the memory regions it has allocated. It also lists what the memory is used for. In our case we examined the COMMAND.COM file which is the command interpreter of the DOS operating system. You can see that the program is stored at segment 01812 but that it also uses memory at 01800 to hold some data and that the environment parameters are stored at segment 01a2d.

Again , this is program only provides you with a starting point. You can now commence the real work and fire up DEBUG.

Chapter 10 : DEBUG a ‘SystemSnooper’

This is probably the most useful tool around on the PC, provided you know how to use it. What’s more is that it comes free with any Microsoft operating system. Alas most people even don’t know that this exists.

You can use this tool to poke around inside the memory of the PC. You can examine files , start programs , disassemble , assemble , and in many ways manipulate everything inside the PC.

Let’s start it up : Open a Dos box and type Debug.

```
C:\>debug
-
```

You will get the debug prompt “-“. You are now in total control of the machine. Feels good doesn’t it ? To display the commands at your disposal simply type ? and press return

```
C:\>debug
-?
assemble     A [address]
compare      C range address
dump         D [range]
enter        E address [list]
fill         F range list
go           G [=address] [addresses]
hex          H value1 value2
input        I port
load         L [address] [drive] [firstsector] [number]
move         M range address
name         N [pathname] [arglist]
output       O port byte
proceed      P [=address] [number]
quit         Q
register     R [register]
search       S range list
trace        T [=address] [value]
unassemble   U [range]
write        W [address] [drive] [firstsector] [number]
allocate expanded memory        XA [#pages]
deallocate expanded memory      XD [handle]
map expanded memory pages       XM [Lpage] [Ppage] [handle]
display expanded memory status  XS
-
```

As you can see , a wealth of instructions allows you to manipulate virtually everything on your computer.

Dump

Probably one of the most important command is the D(ump) command. It allows you to physically examine the contents of the memory

```
-d 0040:0
0040:0000  F8 03 F8 02 E8 03 E8 02-78 03 78 02 00 00 00 00   ........x.x.....
0040:0010  23 C8 00 80 02 00 30 A0-00 00 1E 00 1E 00 75 16   #.....0.......u.
0040:0020  67 22 0D 1C 3F 32 0D 1C-0D 1C 0D 1C 64 20 20 39   g"..?2......d  9
0040:0030  30 52 30 52 34 4B 30 52-3A 34 30 52 0D 1C 01 00   0R0R4K0R:40R....
0040:0040  D6 00 20 00 00 00 00 00-00 03 50 00 40 20 00 00   .. .......P.@ ..
0040:0050  00 31 00 00 00 00 00 00-00 00 00 00 00 00 00 00   .1..............
0040:0060  07 04 00 D4 03 29 30 76-07 87 1C 04 E3 8C 10 00   .....)0v........
0040:0070  00 00 00 00 00 02 08 00-14 14 14 3C 01 01 01 01   ...........<....
-
```

This command allows you to examine physical memory of your machine. Most of the memory locations are in use by either software or data. But some regions contain information provided by the BIOS. There you can find interesting stuff about the machine , like the number of printer ports , the address they are locked on etc.

For more information on these memory fields read the chapter about The DOS system area and The BIOS.

Un-assembling

The second most important command is U(nassemble) Let's take a peek at the COMMAND.COM file

First we retrieve the location of the program using the command "MEM /M COMMAND". This learns us that , in this case , on this computer , it starts in segment 1812. So we fire up debug and execute a dump. And yes , there it is !

```
-d 1812:0
1812:0000  4D 13 18 1A 02 07 FF 07-43 4F 4D 4D 41 4E 44 00   M.......COMMAND.
1812:0010  CD 20 00 A0 00 9A F0 FE-1D F0 A1 01 13 18 BA 26   . .............&
1812:0020  91 FD 47 08 CC 14 13 18-01 01 01 00 02 FF FF FF   ..G.............
1812:0030  FF FF FF FF FF FF FF FF-FF FF FF FF 2E 1A 68 08   ..............h.
1812:0040  13 18 14 00 18 00 13 18-FF FF FF FF 00 00 00 00   ................
1812:0050  07 0A 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1812:0060  CD 21 CB 00 00 00 00 00-00 00 00 00 00 20 20 20   .!...........
1812:0070  20 20 20 20 20 20 20 20-00 00 00 00 00 20 20 20           .....
```

Now we can disassemble this program using the U command. The U command takes as input an offset and a range. The offset is the actual segment we wish to examine . The range is the number of bytes we want to unassemble.

```
-u 1812:0 10
1812:0000 4D            DEC     BP
1812:0001 1318          ADC     BX,[BX+SI]
1812:0003 1A02          SBB     AL,[BP+SI]
1812:0005 07            POP     ES
1812:0006 FF07          INC     WORD PTR [BX]
1812:0008 43            INC     BX
1812:0009 4F            DEC     DI
1812:000A 4D            DEC     BP
1812:000B 4D            DEC     BP
1812:000C 41            INC     CX
1812:000D 4E            DEC     SI
1812:000E 44            INC     SP
1812:000F 00CD          ADD     CH,CL
-
```

Now this doesn’t learn us very much. The reason is that , in our case , we are examining COMMAND.COM , which is somewhat an oddball program .

To really snoop around in programs it is better to load them into memory and then examine them. The easiest way is to simply start debug followed by the filename.

```
C:\>debug textfile
-d
2128:0100  41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50   ABCDEFGHIJKLMNOP
2128:0110  51 52 53 54 55 56 57 58-59 5A 2D 2D 34 00 17 21   QRSTUVWXYZ--4..!
2128:0120  2D 2D 2D 2D 2D 2D 2D 2D-2D 2D 20 20 20 2D 2D 2D   ----------   ---
2128:0130  2D 2D 2D 2D 2D 2D 2D 2D-2D 2D 2D 2D 2D 20 20 20   -------------
2128:0140  2D 2D 2D 2D 2D 2D 2D 2D-2D 2D 2D 2D 2D 2D 2D 2D   ----------------
2128:0150  0D 0A 19 20 20 53 65 67-6D 65 6E 74 20 20 20 20   ...  Segment
2128:0160  20 20 20 20 20 54 6F 74-61 6C 0D 0A 1F 20 20 2D        Total...  -
2128:0170  2D 2D 2D 2D 2D 2D 20 20-20 2D 2D 2D 2D 2D 2D 2D   ------   -------
-
```

As you can see , DEBUG loaded the file in a free part of memory. When we execute the D(ump) command then we immediately see the contents of the file. In this case the file contains the string ‘ABCDEFGHIJKLMNOPQRSTUVWXYZ’

But let’s go back to a real program. Let’s take a closer look at the COMMAND.COM

```
C:\>debug command.com
-d
215C:0100  06 17 BE 1B 02 BF 1B 01-8B CE F7 D9 FC B8 1B 01   ................
215C:0110  06 50 06 B8 18 01 50 CB-F3 A4 CB E9 32 21 7A C9   .P....P.....2!z.
215C:0120  20 00 00 B4 19 00 00 92-16 00 00 B4 1D 00 00 00    ...............
215C:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
215C:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
215C:0150  00 00 00 00 00 00 00 00-E8 6E 00 1E 0E 2E FF 2E   .........n......
215C:0160  1F 01 FB E8 63 00 1E 0E-2E FF 2E 23 01 FB E8 58   ....c......#...X
215C:0170  00 1E 0E 2E FF 2E 27 01-FB E8 4D 00 1E 0E 2E FF   ......'...M.....
-u 215c:0100
215C:0100 06            PUSH    ES
215C:0101 17            POP     SS
215C:0102 BE1B02        MOV     SI,021B
215C:0105 BF1B01        MOV     DI,011B
215C:0108 8BCE          MOV     CX,SI
215C:010A F7D9          NEG     CX
215C:010C FC            CLD
215C:010D B81B01        MOV     AX,011B
215C:0110 06            PUSH    ES
215C:0111 50            PUSH    AX
215C:0112 06            PUSH    ES
215C:0113 B81801        MOV     AX,0118
215C:0116 50            PUSH    AX
215C:0117 CB            RETF
215C:0118 F3            REPZ
215C:0119 A4            MOVSB
215C:011A CB            RETF
215C:011B E93221        JMP     2250
215C:011E 7AC9          JPE     00E9
-
```

This looks completely different then the first time we examined COMMAND.COM. The actual reason is that ,the first time we were only looking at the resident part of COMMAND.COM. What we are seeing no is the complete program before it ever gets executed.

Load

Another very usefull feature is the LOAD command
It allows you to read in a sector from the disk and then examine it.

```
-l 0 2 0 100
-d
2127:0100  CD 10 EB EE BE 82 7D EB-E5 BE 80 7D EB E0 98 CD   ......}....}....
2127:0110  16 5E 1F 66 8F 04 CD 19-BE 81 7D 8B 7D 1A 8D 45   .^.f......}.}..E
2127:0120  FE 8A 4E 0D F7 E1 03 46-FC 13 56 FE B1 04 E8 C1   ..N....F..V.....
2127:0130  00 72 D6 EA 00 02 70 00-B4 42 EB 2D 60 66 6A 00   .r....p..B.-`fj.
2127:0140  52 50 06 53 6A 01 6A 10-8B F4 74 EC 91 92 33 D2   RP.Sj.j...t...3.
2127:0150  F7 76 18 91 F7 76 18 42-87 CA F7 76 1A 8A F2 8A   .v...v.B...v....
2127:0160  E8 C0 CC 02 0A CC B8 01-02 8A 56 24 CD 13 8D 64   ..........V$...d
2127:0170  10 61 72 0A 40 75 01 42-03 5E 0B 49 75 77 C3 03   .ar.@u.B.^.Iuw..
-d
2127:0180  18 01 27 0D 0A 49 6E 76-61 6C 69 64 20 73 79 73   ..'..Invalid sys
2127:0190  74 65 6D 20 64 69 73 6B-FF 0D 0A 44 69 73 6B 20   tem disk...Disk
2127:01A0  49 2F 4F 20 65 72 72 6F-72 FF 0D 0A 52 65 70 6C   I/O error...Repl
2127:01B0  61 63 65 20 74 68 65 20-64 69 73 6B 2C 20 61 6E   ace the disk, an
2127:01C0  64 20 74 68 65 6E 20 70-72 65 73 73 20 61 6E 79   d then press any
2127:01D0  20 6B 65 79 0D 0A 00 00-49 4F 20 20 20 20 20 20    key....IO
2127:01E0  53 59 53 4D 53 44 4F 53-20 20 20 53 59 53 7F 01   SYSMSDOS   SYS..
2127:01F0  00 41 BB 00 07 80 7E 02-0E E9 40 FF 00 00 55 AA   .A....~...@...U.
-
```

In our case we loaded sector starting at the current address (0 = first free memory location) , from disk 2 (C disk ,0 = A , 1 = B , 2 = C , etc …) the first sector of this disk , the first 100hex bytes.

When you dump this you are actually looking at the boot sector of this disk. And as you can see there are the nice strings containing : ‘invalid system disk .. replace the system disk …’ etc.. .

Remember in the BIOS chapter we talked about the booting process of the PC , well there you have it. This boot sector contains a small program that examines itself at then either boots up or displays the message that this disk is not bootable.

Changing memory contents

Besides looking at memory you can also change things in memory . But … Be very very very careful in doing this. You might overwrite part of your operating system and cause your system to crash Nevertheless to show how it works lets fiddle with the video memory.. Besides showing you immediately the outcome of your actions it’s also fun.

To make this work first make sure you have a clear screen and that you are switched to full screen DOS mode (not a windowed DOS box) . you can do this by pressing Alt-enter in windows
Then key in the following commands

```

C:\>cls
C:\>debug
-e b800:0100 48 9f 65 9f 6c 9f 6c 9f 6f 9f
-
```

Fun isn’t it ?

Accessing I/O

Besides snooping around in the memory you can use debug also to look at the hardware IO-ports of the computer.

Debug has 2 commands that allow you to examine the contents of IO ports in your computer

I(nput)
O(utput)

For instance sending something to the printerport would look like this :

```
-
-o 378 ff
-
```

This would put the byte 0xFF onto port 0x378.

```
-i 378
FF
-
```

You can verify this by subsequently inputting the port 0x378

```
-o 378 aa
-i 378
AA
-
```

The above shows a write / read operation the printerport

```
-i 40
F8
-i 40
28
-i 40
2C
-
```

Reading the timer for instance will yield a different number every time

Usage of Debug :

Example 1 : Finding BIOS extensions

You can use the S(earch) command to find sequences of characters in memory . So let’s look for the BIOS extension signature (0x55 0xAA).

```
-s c000:0000 ffff 55 aa
C000:0000
C000:5C6A
C000:6BD0
C000:6BD2
C000:6BD4
C000:6BDE
C000:8000
-
```

You execute the search command starting in at segment 0xC000 and scan the next 0xFFFF bytes (being the complete segment) and look for 0x55 0xAA sequence. This lists a lot of entries. But since we know that BIOS extensions only occur at 2K intervals this eliminates a lot of entries.

Only offsets ending in 0x400 , 0x800 , 0xC000 and 0xF000 can indicate the start of a BIOS extension

```
-d c000:8000
C000:8000  55 AA 04 E9 64 00 00 00-00 00 00 00 00 00 00 00   U...d...........
C000:8010  00 00 00 00 00 00 00 00-4F 00 41 64 61 70 74 65   ........O.Adapte
C000:8020  63 20 42 49 4F 53 3A 41-48 41 2D 32 39 34 30 41   c BIOS:AHA-2940A
C000:8030  55 01 21 43 6F 70 79 72-69 67 68 74 20 41 64 61   U.!Copyright Ada
C000:8040  70 74 65 63 2C 20 49 6E-63 2E 20 31 39 39 36 50   ptec, Inc. 1996P
C000:8050  43 49 52 04 90 78 61 00-00 18 00 00 00 00 01 40   CIR..xa........@
C000:8060  00 00 00 00 80 00 00 00-60 52 FA 50 57 06 BB 00   ........`R.PW...
C000:8070  10 8E C3 66 33 C0 33 FF-B9 00 40 FC F3 66 AF 74   ...f3.3...@..f.t
```

The above learns us that we found the BIOS of an Adaptec SCSI controller board. Examining C000:0000 would yield us the Video extension BIOS

Example 2 using Debug as a Hex file viewer

Sometimes you need a Hex file viewer. Well use Debug .

```
C:\>debug textfile
-d
2128:0100  41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50   ABCDEFGHIJKLMNOP
2128:0110  51 52 53 54 55 56 57 58-59 5A 2D 2D 34 00 17 21   QRSTUVWXYZ--4..!
2128:0120  2D 2D 2D 2D 2D 2D 2D 2D-2D 2D 20 20 20 2D 2D 2D   ----------   ---
2128:0130  2D 2D 2D 2D 2D 2D 2D 2D-2D 2D 2D 2D 2D 20 20 20   -------------
2128:0140  2D 2D 2D 2D 2D 2D 2D 2D-2D 2D 2D 2D 2D 2D 2D 2D   ----------------
2128:0150  0D 0A 19 20 20 53 65 67-6D 65 6E 74 20 20 20 20   ...  Segment
2128:0160  20 20 20 20 20 54 6F 74-61 6C 0D 0A 1F 20 20 2D        Total...  -
2128:0170  2D 2D 2D 2D 2D 2D 20 20-20 2D 2D 2D 2D 2D 2D 2D   ------   -------
-
```

Simply start debug with as argument the filename you wish to examine.
As first command you execute the D(ump) function. This shows you part of the file.

Write down the segment and offset at which the file has been loaded for later reference.
In our case the file has been loaded at 2128:0100

Appendix 1 :Suggested Reading List

The Programmers PC Sourcebook Second edition by Thom Hogan
Microsoft Press ISBN1-55615-321-X

This book is a collection of listings and tables describing every nook and cranny of the IBM PC . From hardware bus connectors to single bit memory location. It even details on the Windows memory usage. The Bible of PC register and memory maps

PC Intern 4 by Michael Tisher
Easy Computing ISBN 09-5157-027-3

Reference book that explains the little known regions of the PC. This provides a good in depth knowledge on how the machine works. This tells the story behind all the registers and interrupts that you can find in the PC sourcebook.

PC Intern 5 by Michael Tisher and Bruno Jennrich
Easy Computing ISBN 90-5167-079-6

While this is the successor of PC intern 4 you really can't live without the old book. This book details more on the Windows environment. A lot of material has vanished (it is still included on the CD ROM that comes with the book) . But when you are on the job it's easier to have NR 4 altogether.

Inside the PC by Peter Norton
Sams Publishing ISBN 0-672-30624-7

A good book that provides a lot of background information about the PC. Includes a lot of sample programs and questionnaires that allow you to evaluate yourself on your knowledge.

The PC inside out by Murray Sargent and Richard L Shoemaker
Addison Wesley ISBN 0-201-62646-2

Provides not only the workings of the PC but gives a crash course on electronics as well. A great book if you're into building your own hardware to fit in to , or attach to the PC. Comes with a nice debugging tool.

The Undocumented PC by Frank van Gilluwe
Addison Wesley No longer available.

This turns the PC inside out and reveals a lot of hidden and little known stuff about the PC. It takes you into uncharted terrain . It details on bugs in BIOS and chip sets. It even points out the BUGS in the CPU's . The accompanying disk contains the most powerful dis-assembler : SOURCER. This is the only tool that every programmer want's ,but will not ask for it. Because he doesn't want you to know that it exists .

A Must for a die-hard assembler programmer

Undocumented DOS by Frank van Gilluwe

Addison Wesley No longer available.

While the other book by this guy turns the PC upside down , this book explores DOS .Every nook and cranny of the operating system is explored and many chapters go into uncharted space. All the secrets that nobody wants to reveal are finally exposed in this book.

A Must for a die-hard assembler programmer

The IBM PC / AT technical Manual

IBM press :PC/AT Technical manual

This book contains all the schematics , BIOS listings ports mapping and everything else about the original PC XT and PC AT. It is the SPEC of the PC. Unfortunately it is not easy to obtain. You have to be a certified IBM developer to get this.

Appendix 2 : VB interaction with the system : WinIO

Most programming languages allow you to access system memory and IO directly. However on today’s 32 bit operating systems this is no longer possible.

The reason is that I/O operations are strictly reserved for a 16 bit process. This means that you need a thunking compiler to provide a connection to the 16 bit DPMI interface which resides inside Windows.

The only compilers which can do that is the Driver / VxD compiler provided by Microsoft. This product comes shipped either in the DDK kit or in the VC++ 5.0 compiler.
Or you have to program in bare 32 bit assembler.

Fortunately a smart guy wrote a toolkit in assembler that allows you to work with the IO without needing complicates stuff like thunk compilers , 16 to 32 bit gateways.

The Manual for this tool can be found in the Win95IO directory.