

> Please send me the IDE interface document. I would be happy to put the
> document on my web page so you can refer everyone to there instead of
> having numerous requests for the document.
>
> Thanks.
>
> Keith Welker
> Ohio University student
> Electrical Engineering Department
> welker@bobcat.ent.ohiou.edu
> welker@homer.ece.ohiou.edu
> <http://www.ent.ohiou.edu/~welker/index.cgi>

I think putting it on your web page would be a good idea. Ever since I posted the announcing message I've been bombed with requests. I never expected this much response.

Well here comes the document:

How to connect an IDE disk to a microcontroller using an 8255
=====

Introduction
=====

Some time ago I have dropped that I had connected an IDE harddisk to one of my microcontrollers. This has provoked a response that I had not foreseen. Since that day I have received some one to two e-mails a day requesting more details about what I had done. At first I have mailed a more or less cryptic decription of my interface to some of the requesters. That only resulted in more e-mail asking for more details. As it seems some people out there are really interested in how my contraption is made. In this description I will attempt to satisfy the information-hunger of all you out there who's appetite I seem to have awakened.

This interface first came to my mind when I re-read some old, old computer magazines. In one of them, the German magazine called C't there was a short description of how and IDE interface is put together. This is in the November issue of 1990. The article describes an IDE interface for both the PC-XT(!) and the PC-AT. The circuit diagrams of the article indcate that the hardware of an IDE interface is in fact very simple. It is essentially a data bus extension from the PC-AT bus to an IDE device. For a PC the hardware comes down to some bus buffers and some decoding. When a disk is connected as an IDE device the PC-AT still 'sees' the old-type control registers of the ancient MFM disk controller. In the article the entire interface is implemented using simple TTL chips. The main problem in a PC seems to be how to keep the harddisk interface and the floppy interface from colliding on some register addresses. If the IDE interface is implemented based on some controller system this is of course no problem.

From a controller point of view an IDE interface could be described as a set of I/O ports. The IDE interface has a 8/16 bits I/O bus, two /CS lines, a /WR and /RD line, three address bits and one interrupt. In this description I assume the most traditional IDE interface. In later IDE interfaces a series of nice so-called PIO modes where added. These PIO modes add things like a ready line, DMA facilities and higher speed data transfers. As you read on you will understand that I only use the so-called PIO mode 0. This is the slowest communication modus on an IDE bus. It is also the easiest one to implement. The data bus

on an IDE interface is used mostly for 8-bits transfers. Only the real disk data reads and writes use the 16-bits bus in full width. You COULD even implement an IDE interface with an 8-bits only data bus. That would mean that you use only half the disk capacity (the lower bytes of the 16-bits-wide bus) but that should work.

When scanning the net I did find an implementation of an IDE interface for 8-bit controllers. This interface was for a (hope I have this correct..) COCO bus. It was implemented in TTL, just like the magazine's interface. The main idea was that whenever a (16-bits) word was read from the IDE bus the upper 8 bits were stored in a latch. The controller could retrieve them from the latch later. Writing to the IDE bus was implemented in the same manner. The IDE bus read/write cycles were in fact simple bus read and write cycles. At first I was about to copy this design. When thinking about it I thought that this TTL design was too complex for what I wanted to do. You need quite some TTL to implement a 16-bits read/write I/O port in TTL on an 8-bits controller.

Hardware description

=====

When implementing a 16-bits I/O port all you need is a bidirectional I/O port and some control bits to generate the /RD, /WR etc... That is when the 8255 came in view. An 8255 has 3 8-bits I/O ports. It can be switched from output to input and back under software control. I used 2 of the 8-bits I/O ports for the data path and use port to generate the IDE control signals. The 74HC04 came into the design later. Once I had the controller and the 8255 strapped together with the IDE connector and a disk I found out that the 8255 has a nasty trait. Whenever you switch the I/O modus of the chip it resets ALL its memory bits. That includes ALL output signals too. For the data bus that is not so much of a problem. The control signals get a real shake when this happens. In particular: The /RESET line of the interface is activated. That makes all control of a disk on this interface impossible (the disk gets a reset at all kinds of odd moments...). I have solved this by simply inverting all the control signals from the 8255 to the IDE bus. When the modus of the 8255 is switched all outputs of the chip go to '0'. That means that all the (low-active) control signals are made inactive by the inversion. That is -in fact- the state where I have them already when I'm about to change the 8255's modus.

At this point I would like to present a nice circuit diagram to show what the contraption I have made looks like. Unfortunately I know of no easy way to do that. This beautiful net is a marvel when it comes down to transporting text, graphics is another matter. A GIF picture would do the work; I do not have any means to produce one. Some schematics drawing package could give a good picture; I have no schematics package and I am not sure what package would be universal enough to be usable by everyone. So I am restricted to a more or less cryptic ASCII description of the hardware. Please, the cryptology is out of need, not out of my liking. Well here it comes:

1) The IDE bus pin connections themselves:

The IDE connector itself is a 40-pins two-row connector:

```

1          39   odd-numbered pins
.....
.....
2          40   even-numbered pins

```

In an IDE bus this connector is used as follows:

pin no:	name:	function:
-----	-----	-----
1	/RESET	All low signal level on this pin will reset all connected devices
2,19,22 24,26,30 40	GND	ground, interconnect them all and tie to controller's ground signal
3,5,7,9,11 13,15,17	D7..D0	low data bus, 3=D7 .. 17=D0. This part of the bus is used for the command and parameter transfer. It is also used for the low byte in 16-bits data transfers.
4,6,8,10 12,14,16,18	D8..D15	high data bus, 4=D8 .. 18=D15. This part of the bus is used only for the 16-bits data transfer.
20	-	This pin is usually missing. It is used to prevent mis-connecting the IDE cable.
21 and 27	/IOREADY	I do not use or connect to this pin. It is there to slow down a controller when it is going too fast for the bus. I do not have that problem...
23	/WR	Write strobe of the bus.
25	/RD	Read strobe of the bus.
28	ALE	Some relic from the XT time. I do not use it, and I'm not the only one...
31	IRQ	Interrupt output from the IDE devices. At this moment I do not use it. This pin could be connected to a controller to generate interrupts when a command is finished. I have an inverter ready for this signal (I need a /IRQ for my controller, an IRQ is of no use to me..)
32	IO16	Used in an AT interface to enable the upper data bus drivers. I do not use this signal. It is redundant anyway, the ATA-3 definition has scrapped it.
34	/PDIAG	Master/slave interface on the IDE bus itself. Leave it alone or suffer master/slave communications problems. Not used (or connected to ANYTHING) by me.
35 33 36	A0 A1 A2	Addresses of the IDE bus. With these you can select which register of the IDE devices you want to communicate.
37	/CS0	The two /CS signals of the IDE bus. Used

- 38 /CS1 in combination with the A0 .. A2 to select
 the register on the IDE device to
 communicate with.
- 39 /ACT A low level on this pin indicates that the
 IDE device is busy. I have connected a LED
 on this pin. The real busy signal for the
 controller I get from the IDE status
 register.

Input/Output status of these signals:

The signals:

A0, A1, A2, /CS0, /CS1, /WR, /RD and /RESET are always outputs
from the controller to the IDE bus.

The signals:

IRQ and /ACT are always outputs from the IDE bus to the
controller (IRQ can be tri-stated by the IDE device when two
devices are connected to the IDE bus.) /ACT can drive a LED (with
resistor of course).

The signals:

D0, D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14,
D15 are bi-directional. They are output from the controller to
the IDE bus when writing, output from the IDE device to the
controller when reading information.

2) The 8255 <-> controller -----

The 8255 is connected to the controller by means of its 8-bit
data bus, the A0 and A1 lines and the /CS, /WR and /RD lines. I
can not give that much info about this. If in doubt: consult the
8255 data sheet. This part depends as much on the controller you
use as anything else. I have the 8255 connected to my controller
(HD63B03R1CP, a Hitachi 6803-derivate..) as an I/O port. Perhaps
some of you have seen my previous e-mails asking for the pinout
of the HD63B03R1CP (52-pins PLCC) chip, well I did find it (Some
department of my work had an old Hitachi databook, voila the
pinout was there). My address decoding puts the 8255 on address
0500H to 0503H in the controller's memory map. That may help if
you decide to try to make sense of my software listing. On this
point you are on your own as to the how to connect a 8255 to your
controller.

3) 8255 <-> IDE connector -----

I have used the 8255's port A to generate the IDE bus control
signals. Some of these control signals pass through an inverter
before I connected them to the IDE connector itself. All of these
signals are always used as output from the 8255 to the IDE bus.
When a control signal is inverted the 8255 pin is connected to
one of the inputs of the 74HC04 and the (corresponding) output of
the 74HC04 is connected to the IDE bus connector.

The ports B and C are used as the 16-bits data bus. There are no
special things in this, it's just a simple interconnection of the
8255 I/O pins to the D0..D15 pins of the IDE connector.

I have connected this as follows:

```

PA.7 --> inverter --> IDE bus /RESET
PA.6 --> inverter --> IDE bus /RD
PA.5 --> inverter --> IDE bus /WR
PA.4 --> inverter --> IDE bus /CS1
PA.3 --> inverter --> IDE bus /CS0
PA.2 -->
           IDE bus A2
PA.1 -->
           IDE bus A1
PA.0 -->
           IDE bus A0

```

```

PB.7 <--> IDE bus D7
PB.6 <--> IDE bus D6
PB.5 <--> IDE bus D5
PB.4 <--> IDE bus D4
PB.3 <--> IDE bus D3
PB.2 <--> IDE bus D2
PB.1 <--> IDE bus D1
PB.0 <--> IDE bus D0

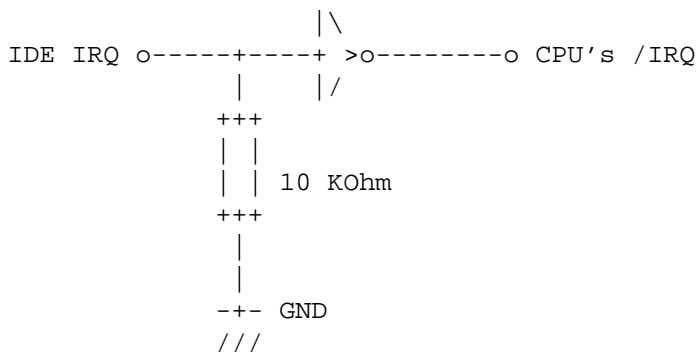
```

```

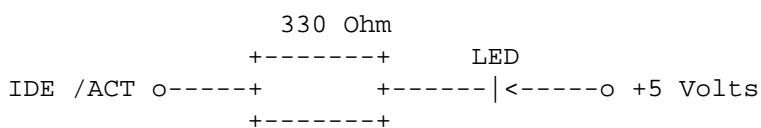
PC.7 <--> IDE bus D15
PC.6 <--> IDE bus D14
PC.5 <--> IDE bus D13
PC.4 <--> IDE bus D12
PC.3 <--> IDE bus D11
PC.2 <--> IDE bus D10
PC.1 <--> IDE bus D9
PC.0 <--> IDE bus D8

```

I have put a 10 KOhm pull-down resistor from the IDE bus IRQ to ground. The IDE IRQ signal is also connected to the input of the (one-remaining) inverter. The output of the inverter is connected to the controller's /IRQ input. As you can see, I do have the hardware for interrupts here, I do not use it. I tried to use it, but got unexplained errors (I probably did something wrong, I have not yet found what)..



The IDE /ACT signal is connected to a 330 Ohm resistor, the other end of the resistor is connected to a LED, the other end of the LED is connected to the +5 Volts. This gives a nice LED indication of when I'm using the disk. This is -as far as I know- the same hardware a PC uses to produce the disk busy LED you may find on the front of a PC box.



So much about the hardware of the IDE interface. I hope that is all clear now. If I have been less than clear, please ask. If I have made mistakes, please complain (I said complain, NOT FLAME!!!).

IDE read/write and register description
=====

The IDE device appears to the IDE bus as a set of registers. The register selection is done with the /CS0, CS1 and A0, A1, A2 lines. Reading/writing is done with the /RD and /WR signals. I have used the 8255 port A signals to make read/write cycles on the IDE bus. What I do is the following:

read cycle:

- 1) put the port B and C of the 8255 in input modus.
- 2) set an address and /CS0 and /CS1 signal on the port A of the 8255.
- 3) activate the /RD of the IDE bus by setting the equivalent bit in the port A of the 8255.
- 4) read the data from port B (and C) of the 8255.
- 5) deactivate the /RD signal on the IDE bus by resetting the equivalent bit of port A of the 8255.

write cycle:

- 1) put the port B and C of the 8255 in output modus.
- 2) set an address and /CS0 and /CS1 signal on the port A of the 8255.
- 3) set a data word (or byte) on port B (and C) of the 8255.
- 4) activate the /WR of the IDE bus by setting the equivalent bit in the port A of the 8255.
- 5) deactivate the /WR signal on the IDE bus by resetting the equivalent bit of port A of the 8255.

The only difference between 8 bits and 16 bits transfers is the following:

- In an 8-bit transfer I use only the port B of the 8255 for data transfer. When writing I put data only on the lower byte of the IDE bus; the upper byte is ignored anyway by the IDE device. When reading I read only port B of the 8255; I never even bother to look at the upper byte.
- In an 16-bit transfer I read/write to both the upper and the lower byte of the IDE bus; thus using both port B and port C.

The IDE device appears as the following registers:

/CS0=0, /CS1=1, A2=A1=A0=0: data I/O register (16-bits)
This is the data I/O register. This is in fact the only 16-bits wide register of the entire IDE interface. It is used for all data block transfers from and to the IDE device.

/CS0=0, /CS1=1, A2..A0=001B: This is the error information register when read; the write precompensation register when

written. I have never bothered with the write precompensation at all, I only read this register when an error is indicated in the IDE status register (see below for the IDE status register).

/CS0=0, /CS1=1, A2..A0=010B: Sector counter register. This register could be used to make multi-sector transfers. You'd have to write the number of sectors to transfer in this register. I use one-sector only transfers; So I'll only write 01H into this register. I do use this register to pass the parameter the timeout for idle modus command via this register.

/CS0=0, /CS1=1, A2..A0=011B: Start sector register. This register holds the start sector of the current track to start reading/writing to. For each transfer I write the start sector in this register. For some fancy reason the sector count starts at 1 and runs up to 256, so writing 00H results in sector number 256. Why that is done is a mystery to me, all other counting in the IDE interface starts at 0.....

/CS0=0, /CS1=1, A2..A0=100B: Low byte of the cylinder number. This register holds low byte of the cylinder number for a disk transfer.

/CS0=0, /CS1=1, A2..A0=101B: High two bits of the cylinder number. The traditional IDE interface allows only cylinder numbers in the range 0..1023. This register gets the two upper bits of this number. I write the cylinder number's upper two bits into this register before each transfer.

/CS0=0, /CS1=1, A2..A0=110B: Head and device select register. The bits 3..0 of this register hold the head number (0..15) for a transfer. The bit 4 is to be written 0 for access to the IDE master device, 1 for access to the IDE slave device. The bits 7..5 are fixed at 101B in the traditional interface.

/CS0=0, /CS1=1, A2..A0=111B: command/status register. When written the IDE device regards the data you write to this register as a command. When read you get the status of the IDE device. Reading his register also clears any interrupts from the IDE device to the controller.

/CS0=1, /CS1=0, A2..A0=110B: 2nd status register/interrupt and reset register. When read this register gives you the same status byte as the primary (/CS0=0, /CS1=1, A2..A0=111B) status register. The only difference is that reading this register does not clear the interrupt from the IDE device when read. When written you can enable/disable the interrupts the IDE device generates; Also you can give a software reset to the IDE device.

/CS0=1, /CS1=0, A2..A0=111B: active status of the IDE device. In this register (read-only) you can find out if the IDE master or slave is currently active and find the currently selected head number. In a PC environment you can also find out if the floppy is currently in the drive. I have not used this register yet.

Some of these registers have bitwise meanings. I'll elaborate on that here:

head and device register:

A write register that sets the master/slave selection and the head number.

bits 3..0: head number [0..15]

bit 4 : master/slave select: 0=master,1=slave
bits 7..5: fixed at 101B. This is in fact the bytes/sector coding. In old (MFM) controllers you could specify if you wanted 128,256,512 or 1024 bytes/sector. In the IDE world only 512 bytes/sector is supported. This bit pattern is a relic from the MFM controllers age. The bit 6 of this pattern could in fact be used to access a disk in LBA modus.

Status register:

Both the primary and secondary status register use the same bit coding. The register is a read register.

bit 0 : error bit. If this bit is set then an error has occurred while executing the latest command. The error status itself is to be found in the error register.
bit 1 : index pulse. Each revolution of the disk this bit is pulsed to '1' once. I have never looked at this bit, I do not even know if that really happens.
bit 2 : ECC bit. if this bit is set then an ECC correction on the data was executed. I ignore this bit.
bit 3 : DRQ bit. If this bit is set then the disk either wants data (disk write) or has data for you (disk read).
bit 4 : SKC bit. Indicates that a seek has been executed with success. I ignore this bit.
bit 5 : WFT bit. indicates a write error has happened. I do not know what to do with this bit here and now. I've never seen it go active.
bit 6 : RDY bit. indicates that the disk has finished its power-up. Wait for this bit to be active before doing anything (except reset) with the disk. I once ignored this bit and was rewarded with a completely unusable disk.
bit 7 : BSY bit. This bit is set when the disk is doing something for you. You have to wait for this bit to clear before you can start giving orders to the disk.

interrupt and reset register:

This register has only two bits that do something (that I know of). It is a write register.

bit 1 : IRQ enable. If this bit is '0' the disk will give and IRQ when it has finished executing a command. When it is '1' the disk will not generate interrupts.
bit 2 : RESET bit. If you pulse this bit to '1' the disk will execute a software reset. The bit is normally '0'. I do not use it because I have full software control of the hardware /RESET line.

Active status register:

This is a read register. I have -up till now- ignored this register. I have only one IDE device (a disk) on my contraption.

bit 0 : master active. If this bit is set then the master IDE device is active.
bit 1 : slave active. If this bit is set then the slave IDE device is active.
bits 5..2: complement of the currently active disk head.
bit 6 : write bit. This bit is set when the device is writing.
bit 7 : in a PC environment this bit indicates if a floppy is present in the floppy drive. Here it has no meaning.

error register:

The error register indicates what went wrong when a command execution results in an error. The fact that an error has occurred is indicated in the status register, the explanation is given in the error register. This is a read register.

- bit 0 : AMNF bit. Indicates that an address mark was not found. What this means I not sure of. I have never seen this happen.
- bit 1 : TKONF bit. When this bit is set the drive was not able to find track 0 of the device. I think you will have to throw away the disk if this happens.
- bit 2 : ABRT bit. This bit is set when you have given an indecent command to the disk. Mostly wrong parameters (wrong head number etc..) cause the disk to respond with error flag in the status bit set and the ABRT bit set. I have gotten this error a lot when I tried to run the disk with interrupts. Something MUST have been wrong with my interface program. I have not (yet) found what.
- bit 3 : MCR bit. indicated that a media change was requested. What that means I do not know. I've ignored this bit till now.
- bit 4 : IDNF bit. Means that a sector ID was not found. I have never seen this happen, I guess it means that you've requested a sector that is not there.
- bit 5 : MC bit. Indicates that the media has been changed. I ignore this bit.
- bit 6 : UNC bit. Indicates that an uncorrectable data error happened. Some read or write errors could provoce this. I have never seen it happen.
- bit 7 : reserved.

Note: I have these registers descriptions from two sources:

- 1) The C't magazine I mentioned before. It's in German, not very complete (the error register description is missing) and very old. It does have a hardware description of and IDE interface however....
- 2) The document X3T10/2008D: Information Technology-AT Attachment-3 Interface (ATA-3), Working draft. This latter document gives an exhaustive overview of the IDE interface. It states more details of the IDE interface than I can ever hope to include in this short description. It does however have one disadvantage: it's BIG. I found the document on the net (on the Western Digital homepage) in the form of an .exe file. When you run this file on a PC you are rewarded with a bigger-than-1 Mbytes .DOC file. That is a Word document. When you print (Wondows-95 has a Word viewer/printer application) it you get a nearly-200-pages paper. If you want to get into the IDE interface seriously I recommend you print this thing.

I hope the description I have given here will allow those that do not have a laser printer and a fast internet link available to get to grips with the IDE bus.

Intermezzo: Disk size limitations on the IDE bus and LBA modus

In the PC world there has been (and still is) a lot of discussion

about 'limits' in the disk interface.

At first (MSDOS versions till 3.3) the disk interface was not able to access more than 32MB on one volume. That was a limitation of the MSDOS file system rather than of the disk interface. The same DOS version that was unable to access bigger partitions than 32MB *WAS* able to access 650MB CDROMs. The limit came from the fact that each disk sector (512 bytes) was registered in the FAT in a 16-bits word. The total partition size was limited by the fact that only 65536 sectors could be addressed. The partition size was thus limited to 65536×512 bytes = 32 MBytes.

Later -as the disks became larger- the disk interface itself ran into its limits. The interface I describe here has room for 16 heads, 256 sectors per track and only 1024 cylinders. With the standard sector size of 512 bytes that leaves you a maximum disk size of $16 \times 256 \times 1024 \times 512 = 2048$ MBytes. That is a real limitation of the IDE interface as I describe it here. It can not access more than some 2 GBytes of disk space.

This was overcome by introducing the so-called LBA modus. In LBA modus the sectors are simply numbered from 0 to -big-. The lowest byte of the LBA sector number is written into the sector number register, the middle 16 bits of the LBA sector number are written in the cylinder number registers (low and high, all 16 bits are used). The highest 4 bits of the LBA sector number are written in the head and device register. That gives you 28 bits of LBA sector number. The sector size was again fixed at 512bytes, so in LBA modus you have access to: $2^{28} \times 512 = 1.37 \text{ E } 11$ (some 137.4 gigabytes) of disk space. This LBA modus has been made mandatory for all new disks (in the ATA-3 spec.) That should keep the disk makers busy for some while to come... If you want to connect a disk larger than 2 GBytes to this IDE interface you too will have to use the LBA modus. How to do that: the bit 6 of the head and device register is set to indicate that LBA modus is used (the fixed pattern of 101B in the bits 7..5 of the head and device register is to be changed into 111B). All other manipulation of the IDE interface is the same for Sector/Head/Cylinder modus and LBA modus.

All other limits in the MSDOS/Windows-whatever disk interface must be due to the BIOS implementation or the file system used. I can find no reason in the IDE definition for 512 MB limits or 8 GB limits at all.

End of intermezzo

IDE registers usage
=====

Ok, now you know what registers the IDE system uses. Next question: How to use them? I do not pretend I have tried every possibility with these registers. As I stated before I have restricted myself to reading/writing data blocks to/from the disk. What to do for that is in fact fairly simple:

- 1) Before doing anything with a device you have to wait till it indicates that it is ready (RDY bit in the status register)
- 2) Next you load the parameters of a command into the appropriate registers. For read/write commands that comes down to writing the cylinder/head/sector numbers into the registers.

- 3) You issue a read or write command.
- 4) You wait till the device signals that it is ready for data transfer (DRQ in the status register).
- 5) Feed the device data (for write) or get the data from the device (for read). In case of a write you could wait for the operation to complete and read the status register to find out what has become of your data.
- 6) Finish!! That's all folks! The IDE interface is a surprisingly simple thing to get to work. If only I had an IDE disk and this kind of information when I was still programming my MSX-computer I'd have had a harddisk connected to it in no time.

IDE commands
 =====

What has been missing in this description till now is the command set. I do not think I can describe the complete command set here. The ATA-3 document is a better source for that than I can give here (All I would be doing is re-entering the ATA-3 document; I have neither the time nor any liking for that). The most usable commands I do intend to describe. Mind: When giving a command you first have to wait for device ready, next put the command parameters in the registers and only then can you give a command (by writing a command byte to the command register). The disk will start executing the command right after you've written the command into the command register.

IDE command:	Description:
-----	-----
1XH	recalibrate the disk. NB: 1XH means that the lower nibble of the command byte is a don't care. All commands 10H..1FH will result in a recalibrate disk command being executed. This command has no parameters. You simply write the command code to the command register and wait for ready status to become active again.
20H	Read sector with retry. NB: 21H = read sector without retry. For this command you have to load the complete circus of cylinder/head/sector first. When the command completes (DRQ goes active) you can read 256 words (16-bits) from the disk's data register.
30H	Write sector (with retry; 31H = without retry). Here too you have to load cylinder/head/sector. Then wait for DRQ to become active. Feed the disk 256 words of data in the data register. Next the disk starts writing. When BSY goes not active you can read the status from the status register.
7XH	Seek. This normally does nothing on modern IDE drives. Modern drives do not position the head if you do not command a read or write.
ECH	Identify drive. This command prepares a buffer (256 words) with information about the drive. If you want the details either look closely at the

interface program I will add at the end of this description or get the ATA-3 document. To use it: simply give the command, wait for DRQ and read the 256 words from the drive. I have found that the modern drives I used give nice information about number of heads, sectors, cylinders etc... One of the disks I tried (a Miniscribe 8051A) gave wrong answers in this buffer. The disk is actually a 4 heads/28 sectors disk. It should be used in a translated modus with 5 heads/17 sectors. In the ident drive response it reported as 4 heads/28 sectors and it will NOT work in that modus. Two other disks (a Quantum 127 MB disk and a Western Digital 212 MB disk) report nicely. If not for the Miniscribe I would use the parameters reported to auto-config the interface to match the disk configuration.

E0H Spins down the drive at once. No parameters. My Miniscribe 8051A does not respond to this command, the other disks do execute this command.

E1H Spins up the drive again. Same remarks as E0H command.

E2H and E3H Auto-power-down the disk. Write in the sector count register the time (5 seconds units) of non-activity after which the disk will spin-down. Write the command to the command register and the disk is set in an auto-power-save modus. The disk will automatically spin-up again when you issue read/write commands. E2H will spin-down, E3H will keep the disk spinning after the command has been given. Example: write 10H in the sector count register, give command E2H and the disk will spin-down after 80 seconds of non-activity. BTW: You can use this command easily on a PC disk too. The harddisk of the computer I am working on now gets this exact command at boot. That saves a lot of noise when I'm typeing long stories like this one.

F2H and F3H The same as E2H and E3H, only the unit in the sector count register is interpreted as 0.1 sec units. I have not tried this command. I think it will work (the E0H/E1H/E2H/E3H commands work, why should this one not work?)

Two-devices considerations =====

The IDE bus is intended for two devices. A master and a slave device. I have not tried anything myself, but the descriptions indicate that it is in fact very simple to connect two devices to the IDE bus. All you have to do is:

- 1) Configure the master/slave jumpers of the devices.
- 2) Select a device before you start giving commands to the devices.

The head and device register has the bit you need to switch from one device to another. You have to write the bit to either 0 for

master or 1 for slave and start controlling the other device.
Mind: BOTH devices will get their registers WRITTEN. Any data or register READ will come from the selected device. ONLY the selected device will execute commands.

Conclusions and ravings
=====

ravings:

This description should be about what you need to connect an IDE disk to any controller. The only thing I have left out are my unsuccessful experiments with the interrupt. What happened there is that I enabled the interrupt, made an interrupt handler that simply read the status register (to get the interrupt to disappear) and re-scheduled the disk interface task. I was rewarded with occasional errors. Some of the read requests got an ABRT error. I do not yet know what I did wrong. I can not have been far off the mark, because most of the commands were executed without comment from the disk. I do intend to try the interrupt modus again later. I theory the interrupt modus should give me a slightly bigger data rate. I have found data rates in the order of 32 KBytes per second when I was using interrupts.

About interrupts: When you want to use the interrupt mechanism all you have to do is enable the interrupt modus of the interface by clearing the interrupt disable bit in the reset and interrupt register. The disk will generate an interrupts as soon as it has completed a command. That means that it will generate an interrupt when it has read a sector from disk (as soon as DRQ gets active) or when it has finished writing a sector to disk. I am not sure about the other commands, but the description says that the disk will generate an interrupt upon the completion of each command.

About the data rate. This IDE interface will never win any award for its speed. It is in one word slow. I get data transfer rates in the order of 25 KBytes per second. I spend most of the time reading/writing data words from/to the disk on a word-by-word basis under software control. On the other hand, my controller has a total (RAM) memory of 24 KBytes. With the current interface I can dump the complete memory to disk in less than one second. That in itself makes me think that for this application the low speed is not so much of an issue.

About the future. I think I will give interrupt modus another try. Not because it gives me a fantastic data rate, I just can't stand it that it does not want to work good.

I am thinking about a proper file system to put on the disk. That will enable me to access the disk in a more normal way that the current 'sea of sectors' approach I use now. A FAT filesystem looks like an absolute horror when combined with a controller, so I'll have to either find something that looks good to me or strap something together myself.

I think I will give the ATAPI command set a try. I have just found and printed the 'SFF committee specification of ATA Packet Interface for CDROMs' (SFF-8020i) document. That document gives a description of how to control an ATAPI CDROM via the IDE bus. On the other hand: I would really not know what to begin with a CDROM connected to a microcontroller. But what the heck, I have no really good idea what to do with a microcontroller with a 40MB

harddisk connected..... I have seen occasional questions in this mailing list about how to control a CDROM via its backside connector: I think this IDE interface with an ATAPI piggy-back software could just do the work.

Till now I've been concentrating on only one side of the interface. The IDE interface is in fact no more than a ready-decoded, buffered interface of a (mostly-PC) hardware system. I could in fact connect god-knows-what at the other end. I have an interrupt, DMA and I/O available, what else could you ask for? A disk or CDROM is only one of the less inspired devices you can connect to an IDE interface. Given the fact that modern PC-motherboards have two IDE interfaces aboard I can think of some nice things I'd like to connect to them. How about some 8 parallel input/output ports, how about a data aquisition system?

Lately I have been able to get a C-mos version of the 8255. My controller system is quite low-power (10 mA for the entire thing). I did not feel good about the IDE extension with a 8255 that used some 80 mA all by itself. The controller system with the D71055C (a C-mos NEC version of the 8255) has gone down to its normal -about 10 mA- power usage. the change from N-mos 8255 to C-mos D71055C has had no other noticable effects than a lower power usage.

Even when I have no disk connected to this contraption it delivers me three 8-bits I/O ports. I am thinking about what other things I could connect to that.

conclusions:

As you can see it is far from difficult to connect an IDE disk to some computer. My implementation with a 8255 and an inverter chip may not be the fastest thing on earth, it works, it's simple and may be usable for many applications where a controller with a large backup store could save the day.

I'll wrap this up with a case-study: The software I use on my 63B03 controller to control the IDE bus. I hope it can clarify any points I have not covered in this description. I know this software to work, I hope it can give someone inspiration to do something similar on some controller system somewhere.

If I have been less than clear, or wrong, or you have some questions left, remarks to make, improvements to suggest, please drop me a line:

paper mail:

Peter Faasse
Hakfort 337
1102 LA Amsterdam
Netherlands

e-mail:

faasse@nlr.nl

Appendix:
=====

The software I use to control a single IDE harddisk using a 63B03


```

0000      |      disk gives only half an answer (and a wrong one
0000      |      as well, it does NOT tell about the 5 heads,17
0000      |      sectors translation it does..) but the other ones
0000      |      like this ident command/display a lot. Also
0000      |      shifted the buffers in memory a bit. I now want
0000      |      to start writing to the disk (till now I have
0000      |      only been reading...).
0000      |      Ok, I really messed up the data on my 42 MB disk,
0000      |      but it DOES write as well as read. I get back
0000      |      what I have written, so far, so good. Now I have
0000      |      to make:
0000
0000      |      1) A proper disk I/O task. That means that I have
0000      |      to implement some way of communicating with
0000      |      the disk I/O task. signals? I REALLY would
0000      |      like to use some sort of mailbox mechanism.
0000      |      For that I will have to extend my scheduler.
0000
0000      |      2) some sort of file-system. I am already
0000      |      contemplating a MFS (Microcontroller File
0000      |      System) for some time now. It's about time to
0000      |      start working on one..
0000
0000      | 18-02-1998 : Started making task # 0E into a disk monitor.
0000
0000      | End-----
0000
0000      | * START INCLUDE *      incl      "debug.inc"      | use all debugger routines etc..
0000      | -----
0000      | include file for the 63B03 debugger
0000      | file : debug.inc
0000      | -----
0000
0000      | -----
0000      | defines for the 63B03 internal I/O
0000      | -----
0000
0000 = 0000      | DdrP1      equ      $0000      | DDR Port 1
0000 = 0001      | DdrP2      equ      $0001      | DDR Port 2
0000 = 0002      | DataP1     equ      $0002      | Data Port1
0000 = 0003      | DataP2     equ      $0003      | Data Port 2
0000 = 0008      | Tcon       equ      $0008      | Timer Control
0000 = 0009      | THigh     equ      $0009      | Timer high byte
0000 = 000A      | TLow      equ      $000A      | Timer low byte
0000 = 0009      | Timer     equ      $0009      | Timer for word access
0000 = 000B      | TComH     equ      $000B      | Timer output compare high byte
0000 = 000C      | TComL     equ      $000C      | Timer output compare low byte
0000 = 000B      | TCom      equ      $000B      | Timer output compare word access
0000 = 000D      | TCapH     equ      $000D      | Timer input capture high byte
0000 = 000E      | TCapL     equ      $000E      | Timer input capture low byte
0000 = 000D      | TCap      equ      $000D      | Timer input capture word address
0000 = 0010      | ModBaud   equ      $0010      | Sio baudrate and mode control register
0000 = 0011      | SioCon    equ      $0011      | Sio control register
0000 = 0012      | SioRec    equ      $0012      | Sio receive register
0000 = 0013      | SioSnd    equ      $0013      | Sio transmit register
0000 = 0014      | RamCon    equ      $0014      | Ram Control register
0000
0000      | -----
0000      | Some of the Interrupts I do not use here and now.
0000      | -----
0000
0000 = 7FF0      | ToiInt    equ      $7FF0      | vector them to RAM
0000 = 7FF4      | IciInt    equ      $7FF4      |
0000 = 7FF8      | IrqInt    equ      $7FF8      |

```



```

0000 = F05A      Step      equ  $F05A      | -> single-step
0000 = F05D      GoBrks  equ  $F05D      | -> GoBreaks
0000 = F060      Trap      equ  $F060      | -> Trap
0000 = F063      Stop      equ  $F063      | -> Stop
0000 = F066      RunInt   equ  $F066      | -> RunInt
0000 = F069      SndRegs  equ  $F069      | -> SndRegs
0000 = F06C      NumByt   equ  $F06C      | -> NumByt
0000 = F06F      UnAsm    equ  $F06F      | -> UnAsm
0000 = F072      DisAsm   equ  $F072      | -> DisAsm
0000 = F075      Tmon     equ  $F075      | -> TMon
0000 = F078      SYS      equ  $F078      | -> SYS
0000 = F07B      SysIrq   equ  $F07B      | -> SysIrq
0000
0000
0000 |-----|
0000 | System calls function numbers |
0000 |-----|
0000
0000 = 0000      sysSusp   equ  $00      | suspend task
0000 = 0001      sysSleep  equ  $01      | sleep for B clock ticks
0000 = 0002      sysPer    equ  $02      | periodic schedule
0000 = 0003      sysSched  equ  $03      | schedule a task
0000 = 0004      sysPar    equ  $04      | get scheduling par
0000 = 0005      sysSusSig equ  $05      | suspend for signal
0000 = 0006      sysSndSig equ  $06      | send a signal
0000 = 0007      sysGetRes equ  $07      | get/suspend on resources
0000 = 0008      sysGivRes equ  $08      | release resources
0000 = 0009      sysSioSnd equ  $09      | send data to sio
0000 = 000A      sysSioRec equ  $0A      | receive/wait for sio byte
0000 = 000B      sysIntWait equ  $0B      | wait for an interrupt
0000 = 000C      sysIntArm equ  $0C      | clear interrupt status
0000
** END INCLUDE **          end
0000
0000 | Defs-----|
0000 |
0000 | The I/O ports of the IDE controller |
0000 |
0000 = 0500      IoBase    equ  $0500      | I/O base address for the disk
0000 = 0500      IoCtl     equ  $0500      | Control byte for IDE
0000 = 0501      IoDatL    equ  $0501      | Low byte data IDE
0000 = 0502      IoDatH    equ  $0502      | High byte data IDE
0000 = 0503      IoMod     equ  $0503      | I/O modus IDE
0000
0000 | The I/O modi I use |
0000 |
0000 = 0080      DMout     equ  $80      | all output
0000 = 008B      DMin      equ  $8B      | ctl = output, data input
0000
0000 | The bits of the control byte |
0000 |
0000 = 0000      DCNOP     equ  %00000000      | Nothing on IDE ctl bus
0000 = 0080      DCRes     equ  %10000000      | /reset bit
0000 = 0040      DCIor     equ  %01000000      | /IORD bit
0000 = 0020      DCIow     equ  %00100000      | /IOWR bit
0000 = 0010      DCCs1     equ  %00010000      | /CS1 bit
0000 = 0008      DCCs0     equ  %00001000      | /CS0 bit
0000 = 0007      DCADR     equ  %00000111      | ADR mask bits
0000
0000 | IDE addresses |
0000 |
0000 = 0007      IDECmd    equ  $07      | adres for command
0000 = 0007      IDESts    equ  $07      | adres status register
0000 = 0006      IDEHd     equ  $06      | adres head number
0000 = 0005      IDECylH   equ  $05      | adres cylinder number high

```

```

0000 = 0004 IDECylL      equ    $04      |  adres cylinder number low
0000 = 0003 IDEsec      equ    $03      |  adres sector number
0000 = 0002 IDEnum     equ    $02      |  adres number of sectors
0000 = 0000 IDEData    equ    $00      |  adres for data bus
0000 = 000E IDERIRQ    equ    $0E      |  adres Reset/IRQ register
0000 = 0001 IDEErr     equ    $01      |  adres Error register
0000
0000 | The head number (0..F) also has the mask for master/slave
0000 | I fix this at Master, I do not think I will use two drives
0000 | on my IDE interface.
0000
0000 = 000F IDEHdA     equ    %00001111 |  head number and mask
0000 = 00A0 IDEHdO     equ    %10100000 |  head number or mask
0000
0000 | The Reset/IRQ register has two interesting bits
0000
0000 = 0100 IDESRes    equ    $00000100 |  Soft Reset bit
0000 = 0010 IDENIRQ    equ    $00000010 |  0 = IRQ active
0000
0000 | The bits from IDESts
0000
0000 = 0080 StsBsy     equ    %10000000 |  busy flag
0000 = 0040 StsRdy     equ    %01000000 |  ready flag
0000 = 0020 StsWft     equ    %00100000 |  Write error
0000 = 0010 StsSKC     equ    %00010000 |  seek complete
0000 = 0008 StsDRQ     equ    %00001000 |  Data Request
0000 = 0004 StsCorr    equ    %00000100 |  ECC executed
0000 = 0002 StsIdx     equ    %00000010 |  Index found
0000 = 0001 StsErr     equ    %00000001 |  error flag
0000
0000 | Command opcodes
0000
0000 = 0010 CmdRecal    equ    $10      |  recalibrate disk
0000 = 0020 CmdRead     equ    $20      |  write a block
0000 = 0030 CmdWrite    equ    $30      |  read block
0000 = 00E0 CmdStop     equ    $E0      |  Stop disk
0000 = 00E1 CmdStrt     equ    $E1      |  Start disk
0000 = 00EC CmdIdent    equ    $EC      |  Identify disk
0000
0000 | The default state (I will always leave it in this state) is:
0000 | IDE bus on input, control word == IDENOP ($FF)
0000
0000 | I use a memory command packet to tell the disk what to do
0000 | This packet has the following makeup:
0000
0000 = 0000 SDA        equ    0          |  offset source/destination
0000 | address in memory for the
0000 | operation
0000 = 0002 LBA3        equ    2          |  offset LBA
0000 = 0003 LBA2        equ    3          |  32-bits number for a disk
0000 = 0004 LBA1        equ    4          |  of max 2.1E12 bytes...
0000 = 0005 LBA0        equ    5          |  should be enough!
0000
0000 | I use Lineair Block Access (LBA) ALL THE TIME. For this disk
0000 | (that does not support it by itself) I have a routine called
0000 | SetLBA to convert the LBA to a CHS configuration.
0000
0000 | I use two parameters to decribe the disk geometry:
0000
0000 | - The number of blocks per cylinder:
0000 |   in CHS terms this is HxS
0000
0000 | - The number of blocks per track
0000 |   in CHS terms this is S

```

```

0000 |
0000 | From these two I can convert LBA to CHS completely. They are
0000 | stored in the following two words:
0000 |
0000 | 42 MB disk:
0000 = 0055 SPC equ 85 | Sectors per Cylinder
0000 = 0011 SPT equ 17 | Sectors per track
0000 |
0000 | 127 MB disk:
0000 SPC equ 272 | Sectors per Cylinder
0000 SPT equ 17 | Sectors per track
0000 |
0000 | 212 MB disk:
0000 SPC equ 420 | Sectors per Cylinder
0000 SPT equ 35 | Sectors per track
0000 |
0000 | I now use the CPU's /IRQ input to handle the disk's IRQ
0000 | signals. The below parameters are for handling these IRQ-s
0000 |
0000 = 0001 irqdisk equ %00000001 | IRQ bit 0 used for the disk
0000 |
0000 | End-----
0000 |
0000 | Debug-----
0000 |
0000 | debug defines for this code
0000 |
0000 | End-----
0000 |
0000 | make the thing as a task
0000 org $4000 | some address
4000 54534B0E db "TSK", $0E | low prio task, a disk is SLOOOW
4004 4040 dw DiskTask | code address
4006 4FFF dw DiskStk | stack
4008 4469736B5461 db "DiskTask" | module name
4010 |
4010 = 4FFF DiskStk equ $4FFF | leave some room (a LOT, in
4010 | fact)
4010 |
4010 | command block 0
4010 4800 CMDBLK0:dw $4800 | SDA = $4800
4012 0000 dw $0000 | LBA = $00000000
4014 0000 dw $0000 |
4016 |
4016 | command block 1
4016 4A00 CMDBLK1:dw $4A00 | SDA = $4A00
4018 0000 dw $0000 | LBA = $00000000
401A 0000 dw $0000 |
401C |
401C org $4040
4040 | start of the real code
4040 DiskTask:
4040 868B ldaa #DMin | control bits output, the rest
4042 B70503 staa IoMod | input
4045 |
4045 | preset control word
4045 8600 ldaa #DCNOP | control word on not active
4047 B70500 staa IoCtl |
404A |
404A | make reset pulse on IDE bus
404A 8680 ldaa #DCRes | make a Reset control word
404C B70500 staa IoCtl | set on output
404F |
404F C601 ldab #1 | spec says 25 us minimum

```

```

4051 8601          ldaa    #sysSleep    | suspend for 10 ms
4053 BDF078        jsr      Sys          |
4056
4056              | set control word to not active again
4056 8600          ldaa    #DCNOP        |
4058 B70500        staa   IoCtl         |
405B
405B BDF00C        disk1:  jsr      Prompt   | give prompt
405E BDF006        disk1:  jsr      RecData  |
4061 8120          cmpa   #' '          | skip spaces
4063 27F9          beq    disk1         |
4065 810D          cmpa   #CR           |
4067 27D7          beq    disktask    |
4069
4069              | let CI do the real work again
4069 CE4072        ldx    #diskttab    |
406C BDF02D        jsr    CI           |
406F 7E405B        jmp    disk1        |
4072
4072              diskttab:
4072 3F             db      '?'          | help
4073 408A          dw      DHelp        |
4075 49            db      'I'          | identify
4076 4126          dw      DIdent       |
4078 52            db      'R'          | Read sector
4079 4130          dw      DRead        |
407B 57            db      'W'          | Write sector
407C 4137          dw      DWrite       |
407E 53            db      'S'          | Stop disk
407F 413E          dw      DStop        |
4081 57            db      'W'          | Start disk
4082 4142          dw      DStart       |
4084 4E            db      'N'          | Reset disk
4085 4146          dw      DReset       |
4087 00            db      $00          | what remains
4088 F04B          dw      NoCmd         |
408A
408A CE4091        DHelp:  ldx    #DHelpTxt    |
408D BDF009        jsr    SndStr        |
4090 39            rts
4091
4091              DHelpTxt:
4091 0D0A4469736B   db      cr,lf,"Disk task commands:"
40A6 0D0A          db      cr,lf
40A8 0D0A49203D20 db      cr,lf,"I = Identify disk"
40BB 0D0A52203D20 db      cr,lf,"R = Read disk sector"
40D1 0D0A57203D20 db      cr,lf,"W = Write disk sector"
40E8 0D0A53203D20 db      cr,lf,"S = Stop the disk"
40FB 0D0A55203D20 db      cr,lf,"U = Wake up the disk"
4111 0D0A4E203D20 db      cr,lf,"N = Reset the disk"
4125 00            db      eom
4126
4126 CE4010        DIdent:  ldx    #CMDBLK0    |
4129 BD43A1        jsr    IdentDisk   |
412C BD43C5        jsr    IdentReport  |
412F 39            rts
4130
4130 CE4010        DRead:   ldx    #CMDBLK0    |
4133 BD432C        jsr    ReadSec     |
4136 39            rts
4137
4137 CE4010        DWrite:  ldx    #CMDBLK0    |
413A BD4361        jsr    WriteSec    |
413D 39            rts

```

```

413E
413E BD4193      DStop: jsr      StopDisk      |
4141 39          rts          |
4142
4142 BD419C      DStart: jsr     StartDisk     |
4145 39          rts          |
4146
4146 BD414A      DReset: jsr     InitIDE       |
4149 39          rts          |
414A
414A
414A |-----|
414A | Routine InitIDE
414A | purp: Init the IDE bus
414A | in  : nothing
414A | out : nothing
414A | uses: nothing
414A
414A | set I/O port modus to activate the control signals
414A 868B      InitIDE: ldaa   #DMin      | control bits output, the rest
414C B70503   staa   IoMod      | input
414F
414F | preset control word
414F 8600      ldaa   #DCNOP      | control word on not active
4151 B70500   staa   IoCtl      |
4154
4154 | make reset pulse on IDE bus
4154 8680      ldaa   #DCRes      | make a Reset control word
4156 B70500   staa   IoCtl      | set on output
4159
4159 C601      ldab   #1          | spec says 25 us minimum
415B 8601      ldaa   #sysSleep   | suspend for 10 ms
415D BDF078   jsr     Sys          |
4160
4160 | set control word to not active again
4160 8600      ldaa   #DCNOP      |
4162 B70500   staa   IoCtl      |
4165
4165 | select the master device, I use a set head number for
4165 | that, The head byte also holds the master/slave select
4165 BD418B     jsr     SetOut      | bus on output
4168 8606     ldaa   #IDEHd      | set address
416A BD41DA   jsr     SetAdr      |
416D 8600     ldaa   #$00      | select head no 0
416F 8AA0     oraa   #IDEHdO  | or byte
4171 BD41C4   jsr     WriteByte  | write to the bus
4174 BD4183   jsr     SetIn       |
4177
4177 | wait till reset executed
4177 BD4212     jsr     WaitNBSY   | wait till disk is ready..
417A          | THEN start giving orders.
417A
417A | give the disk a recalibrate command
417A 8610     ldaa   #CMDRecal   |
417C BD41F1   jsr     WriteCmd   |
417F
417F | wait till command executed
417F BD4212     jsr     WaitNBSY   |
4182
4182 | done
4182 39        rts          |
4183
4183 |-----|
4183 | Routine SetIn
4183 | purp: set all ports of the 8255 for input from the IDE bus

```

```

4183      | in  : nothing
4183      | out : nothing
4183      | uses: nothing
4183      | NB  : This also resets ALL control lines and adr selection
4183
4183 36      SetIn: psha          | save accu
4184 868B      ldaa     #DMin      | get control word
4186 B70503    staa     IoMod     |
4189 32        pula          |
418A 39        rts           |
418B
418B      |-----|
418B      | Routine SetOut
418B      | purp: set all ports of the 8255 for output to the IDE bus
418B      | in  : nothing
418B      | out : nothing
418B      | uses: nothing
418B      | NB  : This also resets ALL control lines and adr selection
418B
418B 36      SetOut: psha          | save accu
418C 8680      ldaa     #DMout     | get control word
418E B70503    staa     IoMod     |
4191 32        pula          |
4192 39        rts           |
4193
4193      |-----|
4193      | Routine StopDisk
4193      | purp: spin down the disk
4193      | in  : nothing
4193      | out : nothing
4193      | uses: nothing
4193
4193      StopDisk:
4193 BD4212      jsr      WaitNBSY   | wait till disk is ready
4196 86E0      ldaa     #CMDStop   | give the command
4198 BD41F1      jsr      WriteCMD  |
419B 39        rts           |
419C
419C      |-----|
419C      | Routine StartDisk
419C      | purp: spin up the disk
419C      | in  : nothing
419C      | out : nothing
419C      | uses: nothing
419C
419C      StartDisk:
419C BD4212      jsr      WaitNBSY   | wait till disk is ready
419F 86E1      ldaa     #CMDStrt   | give the command
41A1 BD41F1      jsr      WriteCMD  |
41A4 39        rts           |
41A5
41A5      |-----|
41A5      | Routine RecalDisk
41A5      | purp: ReCalibrate the disk
41A5      | in  : nothing
41A5      | out : nothing
41A5      | uses: nothing
41A5
41A5      RecalDisk:
41A5 BD4212      jsr      WaitNBSY   | wait till disk is ready
41A8 8610      ldaa     #CMDRecal  | give the command
41AA BD41F1      jsr      WriteCMD  |
41AD 39        rts           |
41AE

```

```

41AE |-----|
41AE | Routine ReadByte
41AE | purp: Read one byte from the IDE bus
41AE | in  : nothing
41AE | out : byte in A
41AE | uses: nothing
41AE | nb  : assumes address and bus have been set
41AE
41AE ReadByte:
41AE 37          pshb          | save b
41AF F60500    ldab          IoCtl    | get current
41B2 CA40      orab          #DCIor   | set Io read
41B4 F70500    stab          IoCtl    | set
41B7 B60501    ldaa         IoDatL   | get data
41BA F60500    ldab          IoCtl    |
41BD C4BF      andb          #low(not DCIor) | reset IoRead again
41BF F70500    stab          IoCtl    |
41C2 33        pulb          |
41C3 39        rts           |
41C4
41C4 |-----|
41C4 | Routine WriteByte
41C4 | purp: Read one byte from the IDE bus
41C4 | in  : byte in A
41C4 | out : nothing
41C4 | uses: nothing
41C4
41C4 WriteByte:
41C4 37          pshb          | save b
41C5 F60500    ldab          IoCtl    | get current
41C8 CA20      orab          #DCIow   | assert Io Write
41CA F70500    stab          IoCtl    |
41CD B70501    staa         IoDatL   | get data
41D0 F60500    ldab          IoCtl    | negate Io Write
41D3 C4DF      andb          #low(not DCIow) |
41D5 F70500    stab          IoCtl    |
41D8 33        pulb          |
41D9 39        rts           |
41DA
41DA |-----|
41DA | Routine SetAdr
41DA | purp: Set an address on the IDE bus
41DA | in  : address in A [0 .. F]
41DA | out : nothing
41DA | uses: nothing
41DA
41DA SetAdr: pshb          | save B
41DB 36        psha          | A too for the moment
41DC
41DC          | set which CS to assert
41DC 8408      anda          %#00001000 | see if adres > 8
41DE 2704      beq          wrad1      |
41E0 8610      ldaa         #DCCs1    | high address
41E2 2002      bra          wrad2      |
41E4 8608      wrad1: ldaa         #DCCs0 | low address
41E6          wrad2:
41E6          | put low bits in place
41E6 33        pulb          | adres -> B
41E7 37        pshb          | back on stack
41E8 C407      andb          #DCAdr    | low adres in B
41EA 1B        aba          | get low adres in A
41EB B70500    staa         IoCtl    | set on control output
41EE
41EE          | get registers back

```

```

41EE 32          pula          |
41EF 33          pulb         |
41F0 39          rts          |
41F1
41F1 |-----|
41F1 | Routine WriteCmd
41F1 | purp: Write one command to the IDE device
41F1 | in  : command in A
41F1 | out : nothing
41F1 | uses: nothing
41F1
41F1 WriteCmd:
41F1 36          psha          | save a
41F2 BD418B     jsr          SetOut | set bus to output
41F5 8607       ldaa         #IDECmd | set address
41F7 BD41DA     jsr          SetAdr  |
41FA 32         pula          | get command byte back
41FB BD41C4     jsr          WriteByte | write the byte
41FE 39         rts          | done
41FF
41FF |-----|
41FF | Routine ReadSts
41FF | purp: get status if the IDE device
41FF | in  : nothing
41FF | out : status byte in A
41FF | uses: nothing
41FF
41FF ReadSts:
41FF BD4183     jsr          SetIn   | set bus to input
4202 8607       ldaa         #IDESTs | set address
4204 BD41DA     jsr          SetAdr  |
4207 BD41AE     jsr          ReadByte | read the byte
420A 39         rts          | done
420B
420B |-----|
420B | Routine ReadReg
420B | purp: get byte from the IDE device
420B | in  : address in A
420B | out : data byte in A
420B | uses: nothing
420B
420B ReadReg:
420B BD41DA     jsr          SetAdr  |
420E BD41AE     jsr          ReadByte |
4211 39         rts          |
4212
4212 |-----|
4212 | Routine WaitNBSY
4212 | purp: wait till the drive indicates ready status
4212 | in  : nothing
4212 | out : nothing
4212 | uses: nothing
4212
4212 WaitNBSY:psha          | save registers
4213 37          pshb         |
4214
4214 | test if device ready
4214 BD41FF     wtrdy1: jsr          ReadSts | get status byte
4217 8480      anda         #StsBsy | get status bits
4219 2709      beq          wtrdy2  |
421B
421B | not ready, wait 10 ms
421B 8601      ldaa         #sysSleep | sleep
421D C601      ldab         #1       | one clock cycle

```

```

421F BDF078          jsr      Sys          |
4222
4222          | try again
4222 20F0          bra      wtrdy1          |
4224
4224          | device is ready
4224 33          wtrdy2: pulb          |
4225 32          pula          |
4226 39          rts          | done
4227
4227          |-----|
4227          | Routine WaitDRDY
4227          | purp: wait till the drive indicates ready status
4227          | in : nothing
4227          | out : nothing
4227          | uses: nothing
4227
4227          WaitDRDY:
4227 36          psha          | save registers
4228 37          pshb          |
4229
4229          | test if device ready
4229 BD41FF        wtdrdy1: jsr      ReadSts          | get status byte
422C 8440          anda      #StsRdy          | get status bits
422E 2609          bne      wtdrdy2          |
4230
4230          | not ready, wait 10 ms
4230 8601          ldaa     #sysSleep          | sleep
4232 C601          ldab     #1          | one clock cycle
4234 BDF078        jsr      Sys          |
4237
4237          | try again
4237 20F0          bra      wtdrdy1          |
4239
4239          | device is ready
4239 33          wtdrdy2: pulb          |
423A 32          pula          |
423B 39          rts          | done
423C
423C          |-----|
423C          | Routine WaitDrq
423C          | purp: wait till the drive indicates ready for data status
423C          | in : nothing
423C          | out : nothing
423C          | uses: nothing
423C
423C          WaitDrq: psha          | save registers
423D 37          pshb          |
423E
423E          | test if device ready for data
423E BD41FF        wtdrq1:  jsr      ReadSts          | get status byte
4241 8408          anda      #StsDrq          | get status bits
4243 2609          bne      wtdrq2          |
4245
4245          | not ready, wait 10 ms
4245 8601          ldaa     #sysSleep          | sleep
4247 C601          ldab     #1          | one clock cycle
4249 BDF078        jsr      Sys          |
424C
424C          | try again
424C 20F0          bra      wtdrq1          |
424E
424E          | device is ready for data
424E 33          wtdrq2:  pulb          |

```

```

424F 32          pula          |
4250 39          rts          | done
4251
4251 |-----|
4251 | Routine ReadBlock
4251 | purp: read one block from the IDE device
4251 | in  : X -> result address (512 bytes)
4251 | out : nothing
4251 | uses: nothing
4251
4251 ReadBlock:
4251 | save registers
4251 36          psha          |
4252 37          pshb          |
4253 3C          pshx          |
4254
4254 | setup for data read
4254 BD4183      jsr      SetIn    | set to input
4257 8600      ldaa     #IDEData  | set address
4259 BD41DA      jsr      SetAdr   |
425C
425C | init loop counter
425C 8600      ldaa     #$00      | 256 word reads
425E
425E F60500      rdblkl: ldab     IoCtl    | get current
4261 CA40      orab     #DCIor    | assert Io Read
4263 F70500      stab     IoCtl    | set
4266
4266 F60501      ldab     IoDatL   | get low byte
4269 E700      stab     0,X      | store
426B 08        inx          |
426C
426C F60502      ldab     IoDatH   | get high data
426F E700      stab     0,X      | store
4271 08        inx          |
4272
4272 F60500      ldab     IoCtl    |
4275 C4BF      andb     #low(not DCIor) | negate Io Read
4277 F70500      stab     IoCtl    |
427A
427A 4A        deca          |
427B 26E1      bne     rdblkl    | loop
427D
427D | done, get things back
427D 38          pulx          |
427E 33          pulb          |
427F 32          pula          |
4280 39          rts          |
4281
4281 |-----|
4281 | Routine WriteBlock
4281 | purp: write one block of data to the IDE device
4281 | in  : X -> data address (512 bytes)
4281 | out : nothing
4281 | uses: nothing
4281
4281 WriteBlock:
4281 | save registers
4281 36          psha          |
4282 37          pshb          |
4283 3C          pshx          |
4284
4284 | setup for data write
4284 BD418B      jsr      SetOut   | bus to output

```

```

4287 8600          ldaa    #IDEData      | set address
4289 BD41DA        jsr     SetAdr          |
428C
428C              | init loop counter
428C 8600          ldaa    #$00          | 256 word write
428E
428E E600          wrblk1: ldab    0,X          | write data byte
4290 F70501        stab    IoDatL         | to I/O ports
4293 08            inx
4294 E600          ldab    0,X          |
4296 F70502        stab    IoDatH         |
4299 08            inx
429A
429A F60500        ldab    IoCtl         | strobe write
429D CA20          orab    #DCIoW        | assert Io Write
429F F70500        stab    IoCtl         |
42A2 C4DF          andb   #low(not DCIoW) | negate Io Write
42A4 F70500        stab    IoCtl         |
42A7
42A7 4A            deca
42A8 26E4          bne    wrblk1         |
42AA
42AA              | done, get things back
42AA 38            pulx
42AB 33            pulb
42AC 32            pula
42AD 39            rts
42AE

```

```

42AE |-----|
42AE | Routine SetLBA
42AE | purp: sets the LBA for the next transfer
42AE | in  : X -> command packet
42AE |       X+0 = ..\ not used or changed here
42AE |       X+1 = ../
42AE |       X+2 = LBA3 (high)
42AE |       X+2 = LBA2
42AE |       X+3 = LBA1
42AE |       X+4 = LBA0 (low)
42AE | out : registers of the IDE controller loaded
42AE | uses: nothing
42AE
42AE | NB: I select ONE sector to be processed
42AE
42AE | I make a stack frame to convert, these are the offsets in
42AE | the stack frame (after tsx, relative to X)
42AE = 0000      LBS3    equ    0          | stack frame LBAS
42AE = 0001      LBS2    equ    1          |
42AE = 0002      LBS1    equ    2          |
42AE = 0003      LBS0    equ    3          | cylinder high
42AE
42AE SetLBA:
42AE | prepare for disk output
42AE BD418B      jsr     SetOut          |
42B1
42B1 | load input data to stack
42B1 3C          pshx          | save X
42B2 A605      ldaa    LBA0,X      |
42B4 36          psha          | 3
42B5 A604      ldaa    LBA1,X      |
42B7 36          psha          | 2
42B8 A603      ldaa    LBA2,X      |
42BA 36          psha          | 1
42BB A602      ldaa    LBA3,X      |
42BD 36          psha          | 0

```

```

42BE 30          tsx          | make X stack frame pointer
42BF
42BF          | init for divide by SPC (=sectors/cylinder)
42BF          | to get the cylinder number. I know the result
42BF          | will fit in 16 bits, so 16 bits result is enough.
42BF C610       ldab      #16          | 16 divide loops
42C1
42C1 37         lbac1:  pshb          | save counter on stack
42C2
42C2          | shift 1 place to left, fill with 0 bit
42C2 6803       asl      LBS0,X      | shift dividend,
42C4 6902       rol      LBS1,X      | fill up with 0
42C6 6901       rol      LBS2,X      |
42C8 6900       rol      LBS3,X      |
42CA
42CA          | test if subtract fits
42CA EC00       ldd      LBS3,X      | get high word
42CC 830055     subd     #SPC          |
42CF 2504       bcs     lbac2         | does not fit, next loop again
42D1
42D1          | fits, set lower cylinder bit, store result
42D1 ED00       std      LBS3,X      |
42D3 6C03       inc     LBS0,X      |
42D5
42D5          | handle loop things
42D5 33         lbac2:  pulb          |
42D6 5A         decb          |
42D7 26E8       bne     lbac1         |
42D9
42D9          | write result to disk registers
42D9 8604       ldaa     #IDECylL      |
42DB BD41DA     jsr     SetAdr         |
42DE A603       ldaa     LBS0,X      |
42E0 BD41C4     jsr     WriteByte     |
42E3 8605       ldaa     #IDECylH      |
42E5 BD41DA     jsr     SetAdr         |
42E8 A602       ldaa     LBS1,X      |
42EA BD41C4     jsr     WriteByte     |
42ED
42ED          | see about head number
42ED 8600       ldaa     #0           | simple subtract will do the
42EF A703       staa    LBS0,X      | work, max H times...
42F1
42F1          | get remaining blocks count
42F1 EC00       ldd     LBS3,X      |
42F3
42F3          | test if one head more
42F3 830011     lbac3:  subd     #SPT          | minus one track's sectors
42F6 2504       bcs     lbac4         | see if result is minus
42F8 6C03       inc     LBS0,X      | no, one more head
42FA 20F7       bra     lbac3         |
42FC
42FC          | gone one too far already
42FC C30011     lbac4:  addd    #SPT          | and sector count
42FF C30001     addd    #$0001        | sector count starts at 1 !!
4302 ED00       std     LBS3,X      | save for a moment
4304
4304          | write to disk registers
4304 8606       ldaa     #IDEHd        | Head is special
4306 BD41DA     jsr     SetAdr         |
4309 A603       ldaa     LBS0,X      |
430B 840F       anda    #IDEHdA       | have to set some bits
430D 8AA0       oraa    #IDEHdO       | to get it working good
430F BD41C4     jsr     WriteByte     |

```

```

4312
4312 8603          ldaa    #IDESec          | sectors is easy again
4314 BD41DA       jsr     SetAdr           |
4317 A601         ldaa    LBS2,X          | low byte only as sector #
4319 BD41C4       jsr     WriteByte        |
431C
431C             | set up for ONE sector transfer
431C 8602          ldaa    #IDEnum          |
431E BD41DA       jsr     SetAdr           |
4321 8601         ldaa    #1             |
4323 BD41C4       jsr     WriteByte        |
4326
4326             | clean up the stack
4326 31            ins             | four bytes LBA
4327 31            ins             |
4328 31            ins             |
4329 31            ins             |
432A
432A             | get X back too
432A 38            pulx            |
432B
432B             | done
432B 39            rts             |
432C
432C             |-----|
432C             | Routine ReadSec
432C             | purp: reads one sector from the disk
432C             | in  : X -> command packet
432C             |         X+0 = dest address high
432C             |         X+1 = dest address low
432C             |         X+2 = LBA3
432C             |         X+3 = LBA2
432C             |         X+4 = LBA1
432C             |         X+5 = LBA0
432C             | out : data read into the destination address
432C             | uses: nothing
432C
432C             | wait for bsy bit cleared
432C BD4212       ReadSec:jsr    WaitNBSY      | wait till disk is ready
432F
432F             | select the device 0
432F 8606          ldaa    #IDEHd          | set address
4331 BD41DA       jsr     SetAdr           |
4334 8600          ldaa    #$00           | set head no = 0
4336 840F         anda    #IDEHdA        |
4338 8AA0         oraa    #IDEHdO        |
433A BD41C4       jsr     WriteByte        |
433D
433D             | load parameters
433D BD42AE       rdsec0: jsr     SetLBA      | set address
4340
4340 8620          ldaa    #CmdRead        | give read command
4342 BD41F1       jsr     WriteCmd        |
4345
4345             | wait for busy gone
4345 BD4212       jsr     WaitNBSY        |
4348
4348             | test on errors
4348 BD41FF       jsr     ReadSts          | get status byte
434B 16           tab             |
434C 8401         anda    #StsErr        | check error bit
434E 2703         beq     rdsec1         | no error -> cont
4350
4350             | error, notify

```

```

4350 BD44D6          jsr      DiskErr          | report the error
4353 17             rdsec1: tba              | test if DRQ
4354 8408           anda     #StsDRQ         |
4356 2601           bne     rdsec2           |
4358
4358                | no data requested, quit
4358 39             rts                |
4359
4359                | transport the data block to destination
4359 3C             rdsec2: pshx             |
435A EE00           ldx     SDA,X             | get dest address
435C BD4251         jsr     ReadBlock        | get the data
435F 38             pulx             |
4360
4360 39             rts                | done
4361
4361 -----
4361 | Routine WriteSec
4361 | purp: Write one sector to the disk
4361 | in  : X -> command packet
4361 |       X+0 = source adres high
4361 |       X+1 = source adres low
4361 |       X+2 = LBA High
4361 |       X+3 = LBA Middle
4361 |       X+4 = LBA low
4361 | out : data written to the disk
4361 | uses: nothing
4361
4361 WriteSec:
4361 BD4212          jsr      WaitNBSY          | wait till disk is ready
4364
4364                | select the device 0
4364 8606           ldaa   #IDEHd           | set address
4366 BD41DA         jsr     SetAdr            |
4369 8600           ldaa   #$00             | set head no = 0
436B 840F           anda   #IDEHdA         |
436D 8AA0           oraa   #IDEHdO         |
436F BD41C4         jsr     WriteByte        |
4372
4372 BD42AE         jsr     SetLBA            | set address
4375
4375 8630           ldaa   #CmdWrite        | give write command
4377 BD41F1         jsr     WriteCmd          |
437A
437A                | wait till command ready
437A BD4212         jsr     WaitNBSY          |
437D
437D                | test on errors
437D BD41FF         jsr     ReadSts          | get status byte
4380 16             tab                |
4381 8401           anda   #StsErr         | check error bit
4383 2703           beq     wrsec1         | no error -> cont
4385
4385                | error, notify
4385 BD44D6          jsr     DiskErr          | report the error
4388 17             wrsec1: tba              | test if DRQ
4389 8408           anda   #StsDRQ         |
438B 2601           bne     wrsec2         |
438D
438D                | no data requested, quit
438D 39             rts                |
438E
438E                | transport the data block to destination
438E 3C             wrsec2: pshx             |

```

```

438F EE00          ldx      SDA,X          | get dest address
4391 BD4281       jsr      WriteBlock     | get the data
4394 38           pulx                     |
4395
4395 39           rts                          | done
4396
4396 BD423C       jsr      WaitDRQ        | wait for data
4399 3C           pshx                     |
439A EE00          ldx      SDA,X          | get dest address
439C BD4281       jsr      WriteBlock     | get data
439F 38           pulx                     |
43A0 39           rts                          |
43A1
43A1
43A1 |-----|
43A1 | Routine IdentDisk
43A1 | purp: Get the disk's parameters from the disk itself
43A1 | in  : X -> command packet
43A1 |       X+0 = dest address high
43A1 |       X+1 = dest address low
43A1 | out : ident packet from the disk in memory
43A1 | uses: nothing
43A1
43A1 IdentDisk:
43A1 BD4212       jsr      WaitNBSY      | wait till disk ready
43A4
43A4 | give command
43A4 86EC         ldaa     #CmdIdent     | send Ident command
43A6 BD41F1       jsr      WriteCmd      |
43A9
43A9 | wait for busy gone
43A9 BD4212       jsr      WaitNBSY      |
43AC
43AC | test on errors
43AC BD41FF       jsr      ReadSts       | get status byte
43AF 16          tab                     |
43B0 8401        anda     #StsErr       | check error bit
43B2 2703        beq      idget1      | no error -> cont
43B4
43B4 | error, notify
43B4 BD44D6       jsr      DiskErr       | report the error
43B7 17          idget1: tba                    | test if DRQ
43B8 8408        anda     #StsDRQ      |
43BA 2601        bne      idget2      |
43BC
43BC | no data requested, quit
43BC 39          rts                          |
43BD
43BD | transport the data block to destination
43BD 3C          idget2: pshx                    |
43BE EE00          ldx      SDA,X          | get dest address
43C0 BD4251       jsr      ReadBlock     | get the data
43C3 38          pulx                     |
43C4
43C4 39          rts                          | done
43C5
43C5 |-----|
43C5 | Routine IdentReport
43C5 | purp: give extensive disk identification report
43C5 | in  : X -> disk command packet
43C5 | out : disk ident info via the sio
43C5 | uses: registers
43C5
43C5 IdentReport:
43C5 | data packet pointed to by the disk command packet

```

```

43C5 36          psha          | save registers
43C6 37          pshb          |
43C7 3C          pshx          | save x too
43C8
43C8            | make X -> ident info NB: this is in LOW-HIGH (Intel)
43C8            | word format. I want to be SOMEWHAT compatible with
43C8            | a PC. Perhaps I can someday read the file-system I make
43C8            | with a PC too...
43C8 EE00        ldx      SDA,X          | x -> ident data
43CA 3C          pshx          |
43CB
43CB            | go dump info
43CB CE445D      ldx      #IDModel      | model name
43CE BDF009      jsr      SndStr          |
43D1 38          pulx         |
43D2 3C          pshx         |
43D3 C636        ldab     #54          | start in SDA block
43D5 3A          abx          |
43D6 C614        ldab     #20          | 20 words
43D8
43D8            | see if any name there
43D8 6D01        tst      1,X          |
43DA 2711        beq      idnm          | zero byte -> no model there
43DC
43DC            | go print the model name
43DC A601        idml:    ldaa     1,X          | first byte (LOW-HIGH)
43DE BDF000      jsr      SndData          |
43E1 A600        ldaa     0,X          |
43E3 BDF000      jsr      SndData          |
43E6 08          inx          |
43E7 08          inx          |
43E8 5A          decb         |
43E9 26F1        bne      idml          |
43EB 2006        bra      idme          |
43ED
43ED            | no name given
43ED CE446D      idnm:    ldx      #IDNoMod      |
43F0 BDF009      jsr      SndStr          |
43F3
43F3            | name done
43F3 01          idme:    nop          |
43F4
43F4            | see about ATA/ATAPI modus
43F4 38          pulx         |
43F5 3C          pshx         |
43F6 CE447B      ldx      #IdAta          |
43F9 BDF009      jsr      SndStr          |
43FC 38          pulx         |
43FD 3C          pshx         |
43FE A601        ldaa     1,X          | get ATA/ATAPI status
4400 8480        anda     #$80          | highest bit
4402 2705        beq      riata          |
4404
4404            | is an ATAPI device
4404 CE4491      ldx      #IdMAtapi      |
4407 2003        bra      israta          |
4409
4409            | is an ATA device
4409 CE448B      riata:   ldx      #IdMata          |
440C
440C            | print device type
440C BDF009      israta:  jsr      SndStr          |
440F
440F            | see about removable/fixed

```

```

440F 38          pulx          |
4410 3C          pshx          |
4411 A600        ldaa          0,X          |
4413 8440        anda          #$40        | seems to be bit 6
4415 2705        beq          rirem        |
4417
4417            | is a fixed disk
4417 CE4499      ldx          #IDFix       |
441A 2003        bra          rifrm        |
441C
441C            | is a removable disk
441C CE449F      rirem: ldx          #IDRem       |
441F BDF009      rifrm: jsr          SndStr       |
4422
4422            | get number of cylinders
4422 CE44A9      ldx          #IDCyl       |
4425 BDF009      jsr          SndStr       |
4428 38          pulx          |
4429 3C          pshx          |
442A A603        ldaa          3,X          |
442C BDF01E      jsr          SndHex       |
442F A602        ldaa          2,X          |
4431 BDF021      jsr          SndHex1      |
4434
4434            | get number of heads
4434 CE44B8      ldx          #IDHead      |
4437 BDF009      jsr          SndStr       |
443A 38          pulx          |
443B 3C          pshx          |
443C A607        ldaa          7,X          |
443E BDF01E      jsr          SndHex       |
4441 A606        ldaa          6,X          |
4443 BDF021      jsr          SndHex1      |
4446
4446            | get number of sectors
4446 CE44C7      ldx          #IDSec       |
4449 BDF009      jsr          SndStr       |
444C 38          pulx          |
444D 3C          pshx          |
444E A60D        ldaa          13,X         |
4450 BDF01E      jsr          SndHex       |
4453 A60C        ldaa          12,X         |
4455 BDF021      jsr          SndHex1      |
4458
4458 38          pulx          | X -> data block
4459 38          pulx          | X -> command packet
445A 33          pulb          | restore registers
445B 32          pula          |
445C 39          rts          | done
445D
445D 0D0A44657669 IDModel:db      cr,lf,"Device name: ",eom
446D 4E6F74207370 IDNoMod:db      "Not specified",eom
447B
447B 0D0A44657669 IDAta: db      cr,lf,"Device type: ",eom
448B 4154412C2000 IDMatata: db      "ATA, ",eom
4491 41544150492C IDMatapi:db      "ATAPI, ",eom
4499 466978656400 IDFix: db      "Fixed",eom
449F 52656D6F7661 IDRem: db      "Removable",eom
44A9
44A9 0D0A43796C69 IDCyl: db      cr,lf,"Cylinders  :",eom
44B8 0D0A48656164 IDHead: db      cr,lf,"Heads      :",eom
44C7 0D0A53656374 IDSec: db      cr,lf,"Sectors   :",eom
44D6
44D6            |-----

```

```

44D6 | Routine DiskErr
44D6 | purp: give extensive error reporting for disk errors
44D6 | in : X -> disk command packet
44D6 | out : error messages about what went wrong with a disk access
44D6 | uses: registers
44D6
44D6 36 DiskErr:psha | save registers
44D7 37 | pshb |
44D8 3C | pshx | save pointer to LBA
44D9
44D9 | general error message
44D9 CE4542 ldx #ErrDsk |
44DC BDF009 jsr SndStr |
44DF
44DF | status
44DF CE4555 ldx #ErrSts | give error string
44E2 BDF009 jsr SndStr |
44E5 8607 ldaa #IDeSts | first the IDE Staus register
44E7 BD420B jsr ReadReg |
44EA BDF01E jsr SndHex |
44ED
44ED | the error bits
44ED CE4563 ldx #ErrBit |
44F0 BDF009 jsr SndStr |
44F3 8601 ldaa #IDEErr |
44F5 BD420B jsr ReadReg |
44F8 BDF01E jsr SndHex |
44FB
44FB | LBA here it happened
44FB CE4571 ldx #ErrLBA |
44FE BDF009 jsr SndStr |
4501 38 pulx |
4502 3C pshx |
4503 EC02 ldd LBA3,X |
4505 BDF01E jsr SndHex |
4508 17 tba |
4509 BDF021 jsr SndHex1 |
450C EC04 ldd LBA1,X |
450E BDF021 jsr SndHex1 |
4511 17 tba |
4512 BDF021 jsr SndHex1 |
4515
4515 | the CHS where it happened
4515 CE457F ldx #ErrCHS |
4518 BDF009 jsr SndStr |
451B 8605 ldaa #IDECylH |
451D BD420B jsr ReadReg |
4520 BDF01E jsr SndHex |
4523 8604 ldaa #IDECYLL |
4525 BD420B jsr ReadReg |
4528 BDF021 jsr SndHex1 |
452B 8606 ldaa #IDEHd |
452D BD420B jsr ReadReg |
4530 BDF01E jsr SndHex |
4533 8603 ldaa #IDEsec |
4535 BD420B jsr ReadReg |
4538 BDF01E jsr SndHex |
453B
453B | end with a new line
453B BDF00F jsr NewLine |
453E
453E | get registers back
453E 38 pulx |
453F 33 pulb |

```

