continuous transfer rates from hard disks, cannot yet be achieved, so there is still a little breathing space in the IDE standard for a few years yet to come.

31.7 SCSI

A very flexible and powerful option for connecting hard disks to a PC is the SCSI (small computer systems interface). The term itself indicates that SCSI is intended for the PC and other small systems (for example, workstations or the Mac). However, the characterization of PCs and workstations as «small» has changed, at least as far as MIPS numbers are concerned, since the Pentium has been on the market. SCSI was derived from the SASI of Shugart Associates (Shugart Associates systems interface). SCSI comes with a somewhat older standard SCSI-I, which is not strict enough in some aspects, resulting in compatibility problems when implementing SCSI-I. The new standard SCSI-II determines the properties more precisely, and additionally defines some more commands and operation modes. SCSI follows a different philosophy to those hard disk interfaces already discussed; this section gives more information on this subject.

31.7.1 SCSI Bus and Connection to the PC

SCSI defines a bus between a maximum of eight units, as well as the protocol for data exchange among them. Such SCSI units may be hard disks, tape drives, optical drives, or any other device that fulfils the SCSI specification. Thus, SCSI drives are intelligent, as are the IDE hard disks; the unit's controller is always integrated on the drive. For connection to the PC a SCSI host adapter is required, which establishes the connection to the PC's system bus similar to the IDE interface. The host adapter itself is also a SCSI unit, so that only seven «free» units remain. Unlike an IDE host adapter, the SCSI host adapter is thus rather complex, as it must recognize all the functions of the SCSI bus and be able to carry them out. But the advantage is that SCSI is not limited to the AT bus. There are also host adapters for EISA or the Mac. The enormous data hansfer rate as well as the high-end performance of the SCSI hard disks doesn't suggest its use in a PC/XT, however. With an accordingly adapted host adapter the same SCSI devices can also be integrated into workstations or an Apple. The Mac has a SCSI interface as standard to connect up to seven external SCSI devices. Apple thus elegantly bypasses its lack of flexibility compared with the IBM-compatible PCs.

Thus the SCSI bus serves only for a data exchange among the SCSI units connected to the bus. A maximum of two units may be active and exchange data at any one time. The data exchange can be carried out between host adapter and a drive, or (as a special feature of SCSI) also between two other SCSI devices (for example, a tape drive and a hard disk). It is remarkable that this data exchange is carried out without the slightest intervention from the CPU; the SCSI drives are intelligent enough to do this on their own. Figure **31.23** shows a scheme of the SCSI bus in the case of integrating a SCSI into a PC.

Every SCSI unit is assigned a SCSI address, which you can set by a jumper on the drive. Addresses in the range 0–7 are valid; according to the SCSI standard, address 7 is reserved for a tape drive. The address is formed by bytes where the least significant bit 0 corresponds to the

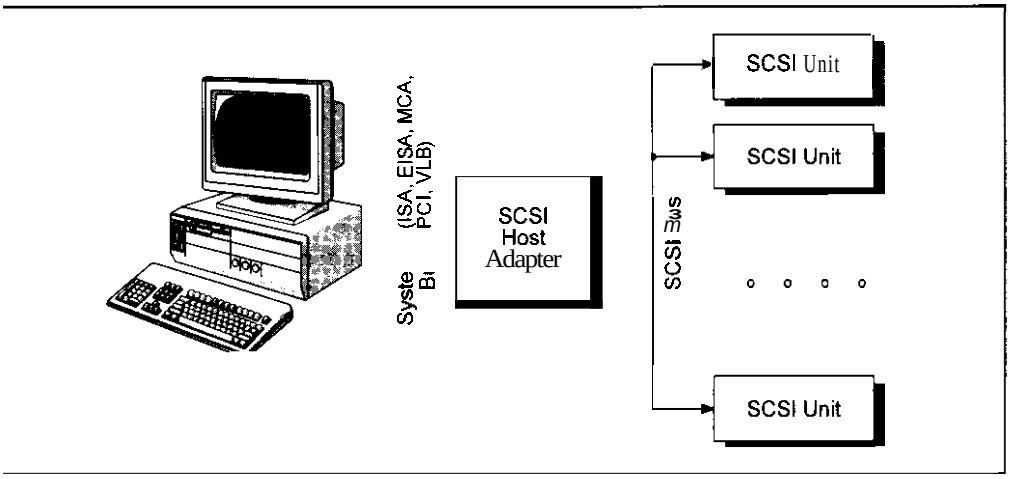


Figure 31.23: SCSI bus and PC *integration. The* SCSI *bus is connected to the* PC *system bus by a* SCSI *host adapter. Up to seven* SCSI *units can* be *served.*

address or SCSI-ID 0, and the most significant bit 7 to the address or SCSI-ID 7. SCSI addresses are transferred via the data section of the SCSI bus (see Table **31.13).**

But don't confuse the SCSI address or SCSI-ID with the logical unit number (LUN). Every target can accommodate up to eight logical units, which you identify in a SCSI command with the **LUN.** An example of this would be a SCSI controller which serves several drives. The controller establishes the connection to the SCSI bus, and further carries out all control functions. Thus the controller is the target. Additionally, the target is assigned several drives (the logical units), which are distinguished by the LUN. Today, external SCSI controllers are rare; most hard disks and also other drives integrate the SCSI controller directly. If you attempt to access such a hard disk you always have to set LUN to a value of 0, as the drive is the first and only logical unit of the target.

Be careful not to cause an address conflict between two drives. The controller of a drive determines its SCSI address at power-up, and then responds to commands that concern this SCSI address. As the host adapter is a SCSl unit, too, with a corresponding SCSI address, several host adapters may access the same SCSl bus. In this way, it is possible for several PCs to share a common SCSI bus, and thus the same drives. They can exchange data via the host adapters without the need for the usual network. Unfortunately, the SCSI bus is restricted to a length of 6 m; a value which is still quite high when compared to the IDE cables with their maximum of about 0.5 m.

The connection between SCSI units is established by means of a 50-wire flat conductor cable with 50-pole plugs. You may also see cables with 25 twisted cable pairs; here one ground line is always twisted around one signal line, similar to Centronics cables. Table 31.13 shows the assignment of the lines and plug pins.

Like all other bus systems for connecting drives (floppy, hard disk, etc.), the SCSI bus must also be terminated by a resistor. This is carried out by removing or disabling the terminating resistor

Signal	Pin	Meaning			
GND	1	ground	TERMPWR	26	termination
DB(0)	2	data bit 0	GND	27	ground
GND	3	ground	GND	28	ground
DB(1)	4	data bit 1	GND	29	ground
GND	5	ground	GND	30	ground
DB(2)	6	data bit 2	GND	31	ground
GND	7	ground	ATN	32	attention
DB(3)	8	data bit 3	GND	33	ground
GND	9	ground	GND	34	ground
DB(4)	10	data bit 4	GND	35	ground
GND	11	ground			
DB(5)	12	data bit 5			
GND	13	ground			
DB(6)	14	data bit 6			
GND	15	ground	BSY	36	busy
DB(7)	16	data bit 7	GND	37	ground
GND	17	ground	ACK	38	acknowledge
DB(P)	18	parity bit	GND	39	ground
GND	19	ground	RST	40	reset
GND	20	ground	GND	41	ground
GND	21	ground	MSG	42	message
GND	22	ground	GND	43	ground
GND	23	ground	SEL	44	select
GND	24	ground	GND	45	ground
	25	-	C/D	46	command/data
			GND	47	ground
			REQ	48	request
			GND	49	ground
			I/O	50	I/O

[1] no connection

*Table 31.13: **SCSI** interface cable layout*

on all but the last drive on the bus. Most SCSI host adapters have a jack for connecting external SCSI devices (typically a scanner, an external SCSI hard disk, or a WORM). Usually the host adapter is one end of the bus and, thus, also incorporates a terminating resistor. If you connect one or more external SCSI devices to the external jack, and if at least one more internal device is present, you must remove or disable the terminating resistor of the host adapter, because now the adapter no longer forms the end of the SCSI bus. Instead, the SCSI bus is guided through the host adapter. Generally speaking, the terminating resistors must always be present at both ends of the bus. Dependent upon the configuration, one of these ends is formed by the host adapter, the last internal SCSI device or the last external device. For the earlier adapters (ST412/506, ESDI and IDE), external devices have not been allowed; there the host adapter or controller always forms one of the two ends of the bus.

As you can see from Table 31.13, eight data bits $\overline{DB(0)}$–$\overline{DB(7)}$ together with one parity bit $\overline{DB(P)}$ and nine control signals are transferred. The SCSI logic generates the parity bit automatically if the unit supports parity; this is not always the case. Using a jumper, you can often determine

whether the parity bit should be generated and checked. All signals are active low. TERMPWR drains surplus charges and damps the SCSI bus. Although the data bus has only eight bits, SCSI is designed for a data transfer rate of up to 4 Mbytes/s in asynchronous and 10 Mbytes/s in synchronous (fast) mode. In asynchronous mode, handshake signals are used for data exchange; in synchronous mode the data transfer is carried out with the handshake signals as clock signals, which leads to a higher transfer rate. But note that not all SCSI units support the synchronous mode. Only in the drives orienting to the SCSI-II standard is the synchronous mode implemented. Moreover, the indicated data transfer rates refer only to the SCSI bus. At which speed the data is passed from or into the PC's main memory via the PC system bus is another question, and it is not determined by the SCSI transfer rates. The overall transfer rate essentially depends upon the quality of the host adapter and the firmware in the adapter's ROM BIOS; a realistic value is 5 Mbytes/s at most. Also, it is decisive, of course, at which speed the data can be read from disk or tape. The principle of the data transfer on a SCSI bus is shown in Figure 31.24.

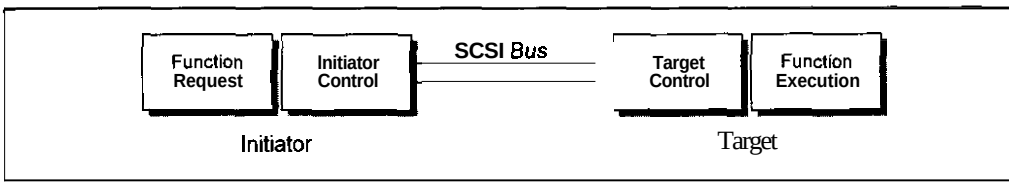


Figure 31.24: Data transfer on the SCSI bus

Any SCSI unit can carry out the function of an ***initiator*** and take control of the SCSI bus by means of control signals. With a SCSI address the initiator activates a certain unit called the target which carries out certain functions. It is of further importance that the initiator occupies the SCSI bus only for command and data transfer, otherwise the bus is free and can be used by other SCSI units. This also applies if the target unit carries out a command (for example, reading a block), and during this time doesn't require a connection to the initiator. Following the command execution, the target unit establishes the connection to the initiator again and transfers the data. The control of the bus is executed by the following control signals:

- $\overline{\text{BSY}}$ (busy): the signal indicates whether the bus is currently busy.
- $\overline{\text{SEL}}$ (select): the signal is used by the initiator to select the target device; on the contrary, the target may also use SEL to re-establish the connection to the initiator after a temporary release of the bus control.
- $\overline{\text{C/D}}$ (control/data): the signal is exclusively controlled by the target, and indicates whether control information or data is present on the SCSI bus. An active signal (with a low level) denotes control information,
- $\overline{\text{I/O}}$ (input/output): the signal is exclusively controlled by the target device, and indicates the direction of the data flow on the data bus relative to the initiator. An active signal (with a low level) means a data transfer to the initiator.
- $\overline{\text{MSG}}$ (message): the signal is activated by the target during the message phase of the SCSI bus.
- $\overline{\text{REQ}}$ (request): the signal is activated by the target unit to indicate the handshake request during the course of a $\overline{\text{REQ}}$/ACK data transfer.

- $\overline{ACK}$ (acknowledge): the signal is activated by the initiator to indicate the handshake acknowledge during the course of a $\overline{REQ}/\overline{ACK}$ data transfer.
- $\overline{ATN}$ (attention): an initiator activates the signal to indicate the attention condition.
- RST (reset): an active signal resets all connected SCSI devices.

You may already recognize that an extensive activation and deactivation procedure with a data transfer by means of handshake signals is carried out. The followingsections discuss the various phases of the SCSI bus, but note that they are only of importance during the course of a data transfer via the SCSI bus. The CPU access to a SCSI device is only affected by this indirectly, as the SCSI logic implemented in the host adapter detects the various bus phases and generates the corresponding control signal automatically. The SCSI bus recognizes eight bus phases in total:

- bus-free
- arbitration
- selection
- reselection
- command
- data
- message
- status.

The last four bus phases (command, data, message and status) together are also called the information transfer phase.

Bus-free Phase

This bus phase indicates that no SCSI unit is currently using and controlling the bus, thus the SCSI bus may be taken over by any connected SCSI unit. This phase is effective when both $\overline{SEL}$ and $\overline{BSY}$ are disabled (high).

Arbitration Phase

In this bus phase a SCSI unit may take control of the bus so that the unit acts as an initiator or target for a bus operation. For this purpose the following procedure is carried out:

- The SCSI bus must be in the bus-free phase as otherwise no unit except the active one is able to take control of the bus; thus $\overline{BSY}$ and $\overline{SEL}$ are both active.
- The unit activates $\overline{BSY}$ and puts its SCSI-ID onto the data bus.
- After a short arbitration delay, the unit investigates the data bus. If another SCSI-ID with a higher priority is active (that is, with a higher SCSI-ID number than its own), then the unit is not allowed to take control of the bus. If this is not the case, then the unit may control the bus; it has won the arbitration and activates $\overline{SEL}$.
- After a short bus-clear delay, the SCSI unit is now able to control the SCSI bus and change the bus signals.

Selection Phase

In this phase an initiator selects a target unit and advises the target to carry out certain functions (for example, reading and writing data blocks). During the selection phase, the $\overline{I/O}$ signal is

inactive to distinguish this phase from the reselection phase of a target. The initiator now outputs the OR-value of its SCSI-ID and the SCSI-ID of the target onto the data bus. Therefore. the two data bits which characterize initiator and target are active.

The thus addressed target must now detect that it has been selected by an initiator and activate $\overline{BSY}$ within a certain time period. If this doesn't happen, then the selection phase has failed and the initiator deactivates $\overline{SEL}$; the SCSI bus enters the bus-free phase. As the initiator has also output its own SCSI-ID besides the target-ID onto the data bus, the target unit is able to identify the initiator. This is important so that the target unit, after a bus-free phase, can activate the correct initiator in the following reselection phase.

Reselection Phase

With the reselection phase a target may re-establish the connection with the original initiator to continue the interrupted operation. This is the case, for example, if a host adapter issues a read command to a target drive. Head positioning and reading the sector concerned takes up to 40 ms, even on fast hard disks; this is a very long time for a computer. Therefore, the target unit releases the SCSI bus and carries out head positioning and reading on its own, but remembers the initiator's SCSI-ID. Thus a bus-free phase occurs, which other devices may use to exchange data. If the target drive has completed the read operation, then it re-establishes the contact to the original initiator by means of a reselection phase and transfers the read data.

The reselection phase proceeds similar to an arbitration and selection phase. The target unit takes over the SCSI bus in an arbitration phase of the SCSI bus, activates BSY, SEL and then the $\overline{I/O}$ signal to identify the phase as a reselection phase and itself as the target. Afterwards, the target outputs its own SCSI-ID and that of the original initiator. The initiator detects that it is selected, and sets up the connection to the target again; now the data exchange can start.

During the course of the four information transfer phases command, data, message and status, phase data and control information are transferred via the data bus. The signals C/D, $1/0$ and $\overline{MSG}$ are used to distinguish the individual transfer phases. If $\overline{I/O}$ is active, then information is transferred from the target to the initiator, otherwise the data transfer proceeds in the opposite direction. Each data transfer in one of these four phases is carried out by handshake. The transmitter puts the data onto the data bus $\overline{DB(0)}$–$\overline{DB(7)}$, and eventually the parity information onto $\overline{DB(P)}$, and activates $\overline{REQ}$ to indicate the validity of the data to the receiver. The receiver fetches the data and activates $\overline{ACK}$ afterwards to inform the transmitter that the data has been accepted. As a result, the transmitter deactivates $\overline{REQ}$. Now the receiver also negates the signal $\overline{ACK}$ (that is, both handshake signals are deactivated), and the next transfer of a data byte by means of a handshake can be carried out.

It is important here that the target unit controls the three signals $\overline{C/D}$, $\overline{1/0}$ and $\overline{MSG}$. The initiator, though, may request a message-out phase by activating $\overline{ATN}$. Table 31.14 shows the connections between the MSG, C/D and T O signals on the one hand, as well as the phase and transfer directions on the other.

$\overline{MSG}$	$\overline{C/D}$	$\overline{I/O}$	Phase	Transfer direction
0	0	0	data-out	initiator→ target
0	0	1	data-in	target+ initiator
0	1	0	command	initiator→ target
0	1	1	status	target* initiator
1	0	0	invalid	–
1	0	1	invalid	–
1	1	0	message-out	initiator→ target
1	1	1	message-in	target→ initiator

Table 31.14: SCSI bus phases

Command Phase

In the command phase the addressed target may request command data from the initiator. For this purpose, the target unit activates the $\overline{C/D}$ signal and deactivates the $\overline{MSG}$ and $\overline{I/O}$ signals. The initiator now transfers the command data.

Data Phase

During the course of the data phase, the target may instruct the initiator to transfer data to the target (the data-out phase), or it can provide data for the initiator (the data-in phase).

Message Phase

In the message phase, the target may advise the initiator to transfer messages to the target (the message-out phase), or it can provide messages for the initiator (the message-in phase).

Status Phase

During the course of the status phase, the target supplies status information to the initiator.

Besides the control signals for issuing the various phases, there are the two further signals, $\overline{ATN}$ (attention) and $\overline{RST}$ (reset). With $\overline{ATN}$ the initiator informs the target that it intends to pass a message. The target fetches the message with a message-out phase. However, the target can transfer a message simply by issuing a message-in phase; only the initiator uses the attention signal. When a SCSI unit activates the $\overline{RST}$ signal, all units are separated from the SCSI bus, all operations are aborted, and the units are set to a defined state.

According to the SCSI specification every initiator implements two sets of three pointers each, called the current pointers and the saved pointers, respectively. The current pointers point to the next command, data and status bytes which are to be transferred between the initiator and the target. They are used by the target currently connected to the initiator. As the connection between initiator and target can be interrupted during an active command and re-established later (reselection phase), the saved pointers are also of further importance. For every active command there is in fact a set of saved pointers, independently of whether the corresponding connection between initiator and target is currently established. The saved command pointer points to the beginning of the command block for the active command, and the saved status pointer to the

beginning of the status area of the active command. The pointers are usually realized by means of registers, which accommodate the corresponding pointer values.

At the beginning of every command the saved data pointer refers to the beginning of the data area until the target unit passes the initiator a message *save data pointer*. Upon this instruction, the initiator shifts the current data pointer into the saved data pointer. Inversely, the target may load the active pointer with the saved pointer by passing the initiator the message *restore pointer*. If a SCSI unit is separated from the bus, only the saved pointers are kept; the active ones are reloaded with new values upon connection with another unit. If the separated SCSI unit is reconnected to the initiator by a reselection phase, then the current pointers are restored from the saved ones. Messages coordinate the connection of the various SCSI units, and pass status information indicating the state of the currently active commands. Thus the protocol of the SCSI bus comprises the physical control signals as well as the logical messages. On the other hand, the commands of the command phase issue certain operations of the SCSI target unit and do not determine the connection of initiator and target. It is only essential for the SCSI units to support the message *command complete* (00h); all other messages are optional.

An initiator informs the target that it also supports the other messages by activating the $\overline{\text{ATN}}$ signal in the course of the selection phase before $\overline{\text{SEL}}$ is activated and $\overline{\text{BSY}}$ is disabled. Then the first message of the initiator to the target after the selection phase is the identification, in the same way as the target must pass the initiator this message after a reselection phase. The SCSI standard defines the following messages:

- Command complete (00h): the target passes the initiator this message in a message-in phase to indicate whether or not a command or a linked command has been completed successfully, and status information has been transferred to the initiator during the course of a status phase. After transferring this message, the target enters the bus-free phase.
- Extended message (01h, xxh): the extended message indicates that further message codes are following. The second xxh byte determines how many bytes are following after xxh. Usually, the first byte after xxh defines the message subcode.
- Save data pointer (02h): the target passes the initiator this message to instruct it to save the current data pointers for the currently connected SCSI unit in the saved data pointers.
- Restore data pointers (03h): the target passes the initiator this message to instruct it to restore the current data pointers from the saved ones.
- Separate (04h): the target passes the initiator this message to indicate that the target is going to interrupt the current connection, that is, to deactivate $\overline{\text{BSY}}$. Later, a new connection is required by means of the reselection phase to complete the command successfully.
- Error at initiator (05h): the message indicates that the initiator has detected an error.
- Abortion (06h): the initiator transfers this message to the target to advise it to abort the current operation, to delete the current data and status information, and to enter the bus-free phase.
- Message rejected (07h): this message can be supplied by the initiator or the target to indicate that the last received message was invalid, or is not implemented in the SCSI unit.
- No operation (08h): the message has no result.

- Message parity error (09h): the message received before shows a parity error.
- Linked command complete (0ah): the target outputs the message to the initiator to indicate that a linked command has been completed, and that status information has been transferred to the initiator during the course of a status phase.
- Linked command with flag complete (0bh): the target outputs the message to the initiator to indicate that a linked command with a set flag has been completed, and that status information has been transferred to the initiator during the course of a status phase.
- Reset bus unit (0ch): the message is passed to the target by the initiator to reset the target.
- Abort tag process (0dh): the message instructs the target to abort the active tag process of a queue. Unlike the message Oeh, here only the currently active process is aborted.
- Clear queue (0eh): this message instructs the target to abort all tag processes of a queue.
- Terminate I/O process (11h): the message instructs the target to terminate (not to abort) the active I/O process as fast as possible.
- Single queue tag (20h): the assigned process can be inserted into the queue by the target at any location. Thus, the target is able to optimize the execution of several initiator requests. The order of such processes can be changed even after the insertion into the queue.
- Tag as queue head (21h): the message instructs the target to insert the assigned process at the beginning of the queue, that is, the process has highest priority and is started as soon as the current process has been terminated or interrupted.
- Ordered queue tag (22h): the assigned process is inserted by the target into the queue and is executed exactly at that location. The order of such processes cannot be changed after the insertion into the queue.
- Ignore wide rest (23h): even if wide mode is active, sometimes the target or initiator does not transfer parameter lists or data as a multiple of two and four bytes, respectively. The receiver must ignore part of the wide SCSI bus. This is achieved by this message.
- Identify (80h to ffh): the message is output by the initiator or target for its own identification. Bit 7 of the message is always set to characterize it as an identification. The remaining seven bits contain the identification code (see Figure 31.25)

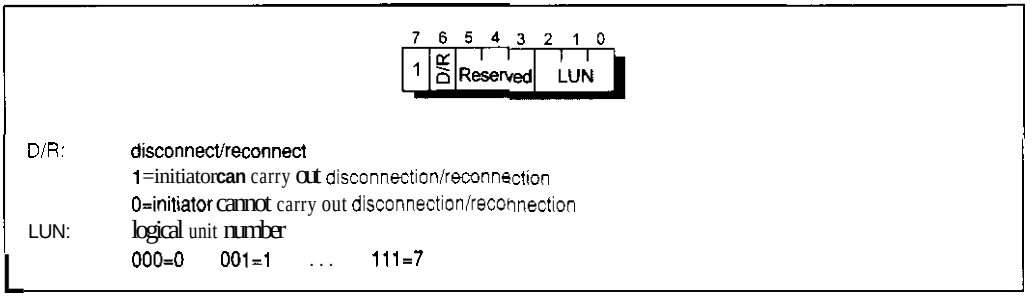
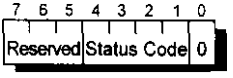


Figure *31.25: Identification code.*

Additional to the listed messages, SCSI-II implements the following extended messages:

- Change data pointer (01h, 05h, 00h): two more bytes follow after the message subcode 00h; they are added to the current data pointer value.
- Request synchronous mode (01h, 03h, 01h): with this message, a SCSI unit is able to indicate that it wants to use synchronous data hansfer. The bytes following after the subcode 01h determine the transfer time (in multiples of 4 ns) and the REQ/ACK delay. Only if the partner also responds with this message is the synchronous mode actually enabled.
- Request wide mode (01h, 02h, 03h): with this message (subcode 03h), a SCSI unit indicates that it wants to use 16- or 32-bit transfer (4th message byte equal to Olh and 02h, respectively). The partner also responds with such a message, and indicates its own width capability. The lesser of the capabilities is used.

Besides the messages the target also transfers a status code to the initiator once a command has been completed. Figure 31.26 shows the structure of this status byte.



Status code: 0000=o.k. 0001=check status 0100=busy
1000=intermediate status/o.k. 1100=reservation conflict

Figure *31.26: Status code.*

The code 0000 indicates that the SCSI unit has executed the command successfully. If a *check status* code is output then an error, an exception, or an abnormal command termination has occurred. You should use the command request sense to determine the cause of this condition. If the target is busy, the status code 0100 is passed. If a linked command is active, the target passes the status code 1000 for every completed individual command except the last one. Thus the status code 1000 confirms the link and takes care that the command sequence is not interrupted. If an error occurs, the status code 0001 is supplied and the linked command is aborted. Finally, a code 1100 means a reservation conflict, that is, a SCSI device has attempted to access a logical unit that is already resewed for another SCSI unit.

Messages are usually not accessible for you as the programmer; they only coordinate the SCSI units among themselves. You can see that SCSI defines a high-level protocol that is not only based on physical signals but also on logical messages. SCSI is therefore very flexible; but presently there is a lack of a strictly defined standard for connecting the SCSI bus to the system bus of a PC. All users of an operating system or an operating system extension (for example, UNIX, OS/2 or Windows) running at least partially in protected mode could tell you a thing or two about that! The problem here is not the SCSI standard, but the appropriate programming of the host adapter to get an access.

The following two sections therefore discuss the programming of the SCSI host adapters ST01 and ST02. With these you may transfer data via the SCSI bus to or from your PC's main memory to access, for example, SCSI hard disks or tape drives.

31.7.2 Programming and Command Phases

The SCSI-II standard defines ten device classes that have quite different commands and functions. As a detailed discussion of all SCSI classes and commands would obviously go beyond the scope of this book (the original literature comprises about 600 pages), the following details mainly the class of hard disks; Appendix H.2 summarizes commands and parameters for that class. In Table 31.15 you will find the ten SCSI-II classes as they are returned by an inquiry command.

Class code	Device
00h	hard disk
01h	tape drive
02h	printer
03h	processor device
04h	WORM
05h	CD-ROM
06h	scanner
07h	optical storage
08h	media changer
09h	communication device
0ah–1fh	resewed

Table 31.15: SCSI-II classes

Unlike previous hard disk interfaces (for example, ST412/506 and IDE), SCSI doesn't deal with tracks and sectors, but regards the complete storage capacity of a hard disk as a continuous list of data blocks with a fixed size. In view of the logical structure, these data blocks are similar to the logical sectors of DOS, which are assigned a logical sector number between 0 and ∞. How a SCSI unit manages this list appearing on the SCSI level internally is the exclusive job of the intelligent controller. On hard disks the logical block number is converted into tracks and sectors, often using zone recording. A SCSI tape drive, on the other hand, can use the logical block numbers in a virtually unaltered form, as on the magnetic tape the data blocks are in succession.

On a PC running under DOS an enormous conversion process is carried out between logical and physical structures:

- the logical DOS sectors must be converted into cylinder, head and sector for INT 13h;
- INT 13h must convert the values for cylinder, head and sector into a logical block number for the SCSI host adapter;
- the addressed hard disk drive converts the logical block number into a value comprising the physical cylinder, head and sector of the SCSI drive.

Thus it is not surprising that every SCSI host adapter has its own BIOS extension to establish the connection between system and SCSI bus, and to carry out the conversion of the physical sectors of INT 13h into logical block numbers for SCSI. The BIOS extension identifies the drive geometry via function 09h of INT 13h to the system. Register addresses and meanings, as well as the programming schemes for the various host adapters, don't coincide in most cases with

any known programming interface, for example the AT task file. Until a short time ago, the lack of a standard for programming the host adapters was a significant deficit of SCSI. Meanwhile, however, two commonly used drivers with a standardized programming interface are available with ASPI and CAM. They, in turn, access the host adapter register in a hardware-dependent manner. On IDE this was never a problem, as the AT task file known from the IBM AT was simply taken.

For a PC with DOS even the lack of a **defined** programming interface to the host adapters was not yet a serious problem, as the host adapters are equipped with an extension ROM for the BIOS routines for INT 13h. The BIOS extension intercepts the INT 13h during the course of the boot process, and replaces the standard routines that are incompatible with the host adapter. But with operating systems running in protected mode (for example, OS/2 or UNIX/XENIX), enormous problems arise. The shipped ROM code is largely only executable in real mode, but cannot support the protected mode operating system. For this reason you always get problems with adapters that have their own BIOS extensions when installing them in a computer running under OS/2 or UNIX/XENIX, if the operating system does not support that adapter on a register level. Only PS/2 machines with advanced BIOS implement a BIOS running in protected mode. Therefore, the BIOS is also loaded from disk, and not just the operating system, if you are using OS/2 on an IBM-compatible PC. If this BIOS doesn't support the installed adapter (and this applies not only to hard disk controllers but to all other adapters, too), then the adapter is not operating correctly, or the BIOS extension prevents you from working in protected mode.

Also, the lack of a defined programming interface to the host adapter means that instead of the ROM code the operating system cannot be equipped with a universal driver, which can access all SCSI drives according to the SCSI specification. As long as the boot routine is running in real mode, the system data can be read from the SCSI drive to boot the computer. But at that moment when the initialization routine switches the processor into protected mode, the hard disk becomes «dead» as the real-mode BIOS denies the access when the hard disk is addressed, or hangs up the PC. Such problems cannot occur with IDE or ESDI hard disks, as they are programmed via registers. These registers, though, are located in the I/O address space for which even in protected mode no address transformation is carried out: only the access rights are checked by the operating system. Therefore, the system can manage its own accesses, and problems hardly ever arise.

At the end of every command the target (that is, the drive) returns a status byte to the initiator (the host adapter). If this byte indicates an error condition, then you should investigate the cause using the request sense command. With a single byte the cause cannot be described exactly enough, and the SCSI host adapter doesn't implement a status register with error information, as was the case, for example, for IDE.

The SCSI commands follow a strict scheme. The first command byte always contains the command code, the second the number of the addressed target, and the third a control byte. SCSI commands always have six, ten or twelve bytes. The 10- and 12-byte commands are denoted as extended. The command code is divided into a 3-bit group code and a 5-bit code with the command within that group (see Figure 31.27).

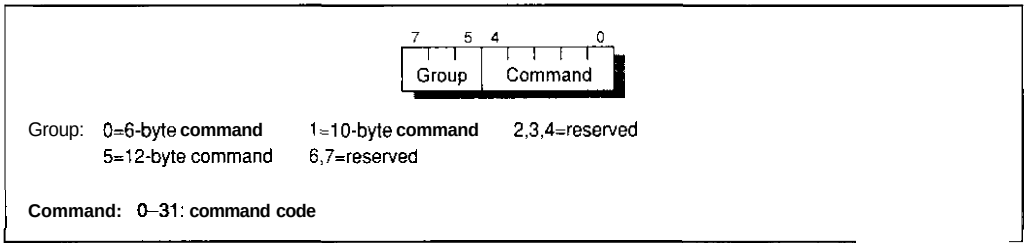


Figure 31.27: Command byte structure.

The structure of the command byte gives rise to eight different command groups; only four of them (0, 1, 2 and 5) are currently in use. The five command bits allow 32 commands per group; thus a maximum of 256 different commands is possible. You must set all reserved bits of the command codes to a value of 0. Figure 31.28 shows their structure.

The first byte of the command block represents the command code, consisting of the 3-bit group and the 5-bit command. The logical unit number (*LUN*) specifies the address of the logical unit to access within the target. For drives with an embedded controller this is always equal to 0. The logical block address (*LBA*) has 21 bits for a 6-byte command and 32 bits for a 10-byte or 12-byte command. It indicates the number of the intended data block. Thus, with a 6-byte command you can access a maximum of 2M blocks and with a 10-byte or 12-byte command a maximum of 4G blocks. For hard disks where one block usually corresponds to a 512-byte sector, a 6-byte

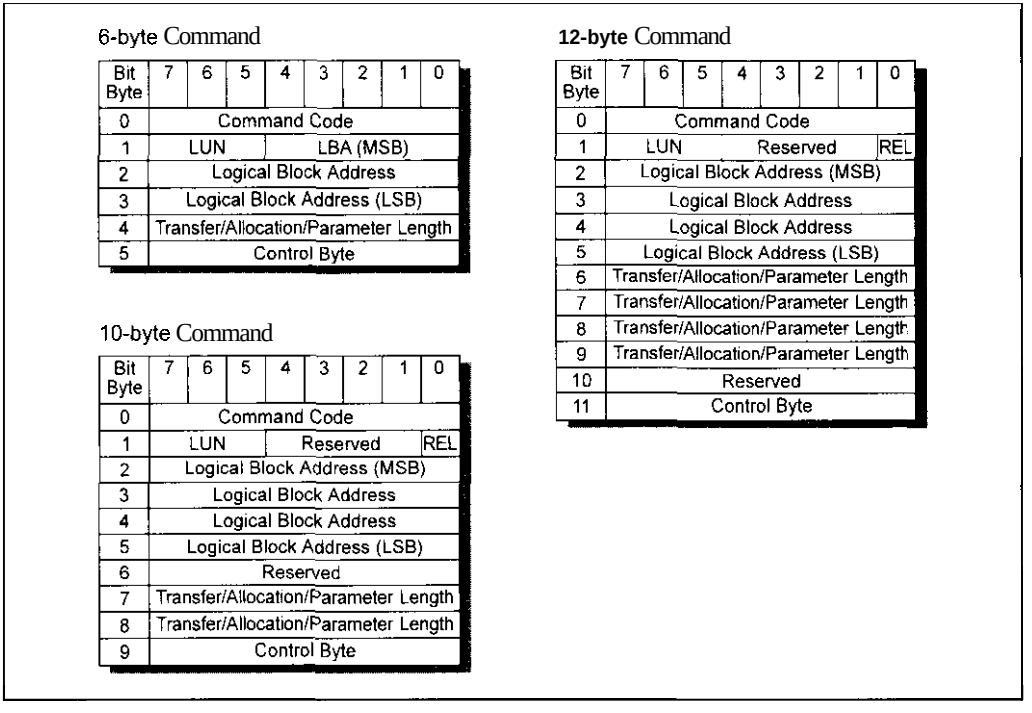


Figure 31.28: Structure of the 6-, 10- and 12-byte commands

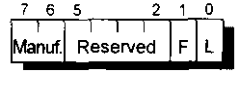
command is sufficient in most cases. You can then access 1 Gbyte of data. If you set the REL bit for relative addressing then the block address is relative to the block referred to in the previous process. The block address is interpreted in this case as the 2'complement of a signed number. Most SCSI units and hard disks, though, don't support relative addressing. Note that this option is missing on 6-byte commands.

The *transfer* length specifies the amount of data to be transferred. Usually, this is the number of intended blocks, but some commands olso use the number of bytes. Details on this subject are discussed in Appendix H, together with a list of all SCSI commands. Six-byte commands with a byte for the transfer length allow the transfer of a maximum 256 blocks, where a value of 0 means that 256 blocks are transferred. Thus you can read or write up to 256 blocks all at once, for example. On an IDE interface this means a multisector transfer. With the use of a 10-byte command, which reserves two bytes for the transfer length, up to 65 535 blocks may be transferred with a single command. A value of 0 here really means that no block is transferred. Even larger is that amount with a 12-byte command; here, four bytes are reserved for the transfer length, corresponding to 4G data blocks. Thus, SCSI commands are very powerful; the 65 535 sector blocks of a 10-byte command with 512 bytes each corresponds to nearly 32 Mbytes of data, after all! Thus you may transfer, for example, a complete DOS partition (before version 3.30) by means of a single SCSI command. The 4G data blocks with 512 bytes each should also be enough for the future graphics interfaces according to SCSI-III.

The parameter list entry usually indicates the number of bytes transferred during the data-out phase of a command as a parameter list to the target. This applies, for example, to the mode select command.

If you issue a command used for returning sense data such as request sense or inquiry, then you need to enter in the allocation length field the number of bytes that the initiator is to receive from the target. The target terminates the transfer of the sense data when the value indicated in the field allocation length is reached. If the value is higher than the number of transferred sense bytes, then the target terminates the data-in phase earlier. The remaining bytes are not defined.

Every command block is terminated by a control byte, which mainly controls the linking of several commands. Figure 31.29 shows its format.



Manuf: manufacturer (insignificant)
reserved: 0
F: flag
1=message "linked command with flag complete" output
1=message "linked command complete" output
L: link
1=command link 0=no command link

Figure **31.29:** *Control byte structure.*

The two most significant bits *manufac* are available for the manufacturers of SCSl units; their value is ignored by the target. The four reserved bits must be set to 0. If the link bit L is set, then the initiator requests a command link. The target then returns only an intermediate status after completing a command, and requests the next command. The connection is not interrupted (as is the case for individual commands) and re-established by means of a bus-free and a selection phase, but remains effective so that the initiator can pass the next command block at once. Closely related to the link bit is the flag bit F. If L is cleared then F should also be equal to 0. With an enabled link a cleared flag bit F indicates that the target passes the initiator a message *linked command complete* after successfully completing a partial command. If the flag bit is set then a message *linked command with flag complete* is transferred instead.

You start a command execution by transferring a 6-, 10- or 12-byte command block to the target. Some commands additionally require a parameter list, which is hansferred during the course of a data-out phase. If you employ the ST01 then you must therefore proceed as follows:

- Clear the SCSI ENAB bit in the control register and set SCSl ENAB to activate the SCSI bus; write the SCSI-ID of the host adapter into the SCSI data port, and start an arbitration phase of the host adapter by means of the ARB STRT bit in the control register.
- Observe the ARB CMPL bit in the status register to determine the end of the arbitration phase, and to confirm that the host adapter has gained control of the SCSI bus.
- Issue a selection phase to select the intended target by activating the SEL line with the SCSI SEL bit of the control register, and by outputting the target's SCSI-ID via the SCSI data port.
- Observe the SCSI BSY bit in the status register afterwards to determine whether the addressed target has responded to the selection phase and has taken control of the bus.
- Observe the SCSl C/D, SCSI MSG and SCSl 1/0 bits in the status register to determine the beginning of a command phase; for this purpose, SCSl MSG and SCSI 1/0 must be cleared, but SCSI C/D must be set.
- Transfer the six or ten command bytes of the command block via the SCSI data port to the target to start command execution.
- If, additionally, a parameter list has to be transferred for the command concerned (mode select, for example), then observe the bits SCSI MSG, SCSI C/D and SCSI 1/0 to determine the beginning of a data-out phase (SCSI MSG, SCSI C/D and SCSI 1/0 cleared), and provide the parameter list via the SCSI data port.
- If a command (write block, for example) requires the transfer of a data block, then observe the SCSI MSG, SCSI C/D and SCSI I/O bits to determine the beginning of a data-out phase (SCSI MSG, SCSI C/D and SCSI 1/0 cleared), and provide the data block via the SCSI data port.
- If a command (read block, for example) returns a data block to the initiator (that is, the host adapter), then observe the bits SCSI MSG, SCSl C/D and SCSI 1/0 to determine the beginning of a data-in phase with cleared bits SCSI MSG, SCSI C/D and set bit SCSI I/O, and to fetch the data block via the SCSI data port.

- Observe SCSI MSG, SCSI C/D and SCSI I/O bits in the status register to determine the beginning of a status phase when SCSI MSG is cleared and SCSI C/D and SCSI I/O are set; the status byte is read via the SCSl data port.
- Observe the SCSI MSG, SCSl C/D and SCSI 1/0 bits in the status register to determine the beginning of a message-in phase (SCSI MSG, SCSI C/D and SCSl I/O are set), and to fetch the message via the SCSI data port.

You can see that the data exchange via host adapter and the SCSI bus is a rather extensive operation. All the details concerning correct programming of the host adapters would go far beyond the scope of this book. I have no other choice than to refer the SCSl freaks among you to the original literature on this subject. I hope, nevertheless, that the basic concepts and idea behind the very powerful and flexible SCSI have become somewhat clearer now. Consider that you can, by means of SCSI, integrate hard disks, tape drives, optical drives, and all the other revolutions that will surely arise in the future into your PC without the need of a hardware adapter or memory-eating driver for every single device. A general interface standard gives rise to an enormous flexibility but the concrete programming, unfortunately, becomes somewhat ponderous.

More powerful SCSI host adapters than the ST01 and ST02 implement a major portion of the SCSI bus control, which has to be programmed explicitly on an ST01 or ST02, by means of a processor, which is in turn controlled by firmware or exclusively by means of hardware, that is, an ASIC. The data exchange between host adapter and drive is then carried out very quickly. EISA SCSI host adapters often transfer the data from the host adapter by means of an EISA DMA channel operating in burst DMA mode C at the full width of 32 bits. Such SCSI systems achieve the highest transfer rates of up to 10 Mbytes/s if the host adapter has an on-board cache. But the 10 Mbytes/s refers only to the pure DMA transfer, that is, the path host–adapter–cache to main memory, or vice versa. The Transfer rates on the SCSI bus are significantly lower but, nevertheless, impressive (up to 4 Mbytes/s).

31.7.3 ASPI

As already explained, the greatest problem concerning SCSI host adapters is the lack of a binding standard for a programming interface. With its help, it should be possible to send SCSI instructions, data and parameters to a SCSI target, or to receive data and parameters from a target, in a simple way. In the meantime, however, two de facto standards have been produced, namely the *Common* Access *Method* (CAM) from ANSI and the Advanced SCSI *Programming* Interface (*ASPI*) from Adaptec. In the following section, I would briefly like to introduce the basics of ASPI.

Normally, ASPI is a driver which is loaded during start-up of your PC. You can use the ASPI functions (Table 31.16) as they are stored in the stack segment and offset (in this sequence) of a SCSI Request Block (*SRB*). Before the call, the SRB contains all other important information for the requested function and, after the request has been performed, it contains the applicable data from ASPI or the SCSI target. The structure of the first eight SRB bytes is the same for all ASPI functions. Then, you must perform a far call to the entry point of the ASPI driver. In DOS you

Code	Function
00h	Host adapter inquiry
01h	Determlne unit type
02h	Perform SCSI Instruction
03h	Interrupt SCSl instruction
04h	Reset SCSI unit
05h	Set parameters for host adapter
06h	Reserved for target mode
07h–7fh	Reserved for expansions
80h–ffh	Reserved for manufacturer

Table 31.16: The ASPI functions

can determine this entry point using INT 21h, function 44h (communication with device drivers), sub-function 02h (read control code). For this you must first open the ASPI driver with the help of the ASPI driver *SCSIMGR$* and use the returned DOS handle as the AL value for INT 21h, function 4402h.

As an example, I would like to explain the usage of the *perform SCSI instruction* ASPI function. In Table 31.17 you can see the layout of the SRB for this function. From the complete SRB for this ASPI function, bytes 00h, 02–17h, 1ah–40h+l2–1 are the input values, and the bytes Olh, 18h–19h, 40h+l2–40h+l2+l1–1 are the returned values. The completion of the ASPI function call is identified by a status value other than 00h (byte 01h). In Table 31.18 you will find all the valid status values for the ASPI functions.

Byte	Content
00h	02h (perform SCSI instruction)
01h	Status
02h	Host adapter number
03h	Bit 0 POST, Bit 1: link, Bit 2.3: direction, Bit 3.5: resewed
04h–07h	Reserved
08h	ID of target
09h	LUN
Oah-Odh	Size of the data buffer
Oeh	Extent of sense data in byte (l1)
0fh–12h	Segment:Offset of the data buffer
13h–16h	Segment:Offset of the SRB link
17h	Extent of SCSI instruction in byte (l2)
18h	Status of host adapter
19h	Target status
lah–ldh	Segment:Offset of POST
1eh–3fh	Reserved
40h–40h+l2–1	SCSI instruction
40h+l2..40h+l2+l1–1	Sense data

***Table** 31.17: Layout of the SRB for the ASP1 function perform SCSI instruction*

To call up the ASPI function with the required SCSI instruction, you must first prepare a buffer for the SRB itself, and then a sufficiently large data buffer (for example, if you wish to read or

Status value	Meaning
00h	Function is currently being performed (ASPI in progress)
01h	Function successfuily completed
02h	Function interrupted by host instruction
03h	SCSI command cannot be interrupted
04h	General failure
80h	Invalid SRB format
81h	Invalid host adapter specified
82h	SCSI target not found

Table 31.18: ***Status value*** *of the ASPI functions*

write a disk drive block). Finally, the SRB fields are initialized (you will find the meaning of the entries in Appendix H.3). The call is then simply accomplished in the following way (address of the SRB: SRB_seg:SRB_off, jump address from ASPI: ASPI_ptr = ASPI_seg:ASPI_off):

```
ASPI_start:          ; ASPI function call
mov ax, SRB_seg      ; Load segment from SRB into accumulator ax
push ax              ; Put segment value on the stack
mov ax, SRB_off      ; Load offset from SRB into accumulator ax
push ax              ; Put offset value on the stack
call ASP_ptr         ; Par call to the ASPI jump point
pop ax               ; Get SRB offset from the stack
pop ax               ; Get SRB segment From the stack

SRB_check:           ; Check whether ASPI function has been completed
mov bx, SRB_seg      ; Load segment from SRB into bx
mov es, ax           ; Lord segment from SRB into extra segment es
mov bx, SRB_off      ; Load offset from SRB into bx
check-start:         ; Begin the checking
mov al, [es:] [bx+1] ; Load status byte in SRB info al
or al, 00h           ; Compare al with valve 00h
JZ check-start       ; check again if al equals zero
```

SRB_check is used to interrogate the status byte in the SRB until a value other than 0 occurs. This is the indication that the requested SCSI function has been performed. The status byte in the SRB then contains an end code (see Appendix H.3), the sense data in the SRB (byte 40h+l2..40h+l2+l1–1) and the sense code from the SCSI adapter.

ASPI supports SCSI instruction chains by «linking» a number of SRBs. The 4 bytes *Segment:Offset* of the *SRB* link (Bytes13h..16h) entry are used for this. It refers to the SRB of the next instruction in a chain. To actually activate the SRB linking, you must set the link bit in byte 3. This applies to all except the last SRB in a chain. As soon as an addressed target sends back the linked command complete status code, execution of the next instruction in the chain commences, that is, the following SRB is performed.

If you set the POST bit 0 in the SRB, ASPI carries out a so-called Posting and, after execution of the SCSI instruction, jumps to the far address stored in bytes 1ah–1dh. You can use the call in addition to checking the status code in SRB, for example, as an alternative method of identifying the completion of an ASPI function. A further use would be the processing of SCSI

results, such as the evaluation of sense data. Immediately following the call-up of the POST routine, the 4 byte return jump address (Segment:Offset) for ASPI, and a 4 byte SRB pointer (Segment:Offset) for the SRB which is currently being performed are stored on the stack.

31.7.4 Other Standardized SCSI Programming Interfaces

For a complete picture, 1 would like to introduce two additional standardized programming interfaces for access to the SCSI bus: the *Common* Access ***Method*** (CAM) and TARGA. Owing to the lack of space (a detailed specification for ASPI, CAM and TARGA could quite easily fill a hook of its own), I will have to limit the explanation to those functions that are implemented and a few general details.

Common Access Method (CAM)

Like ASPI, the CAM operates with the help of a data structure in the memory, the CAM *Control Block* (*CCB*), corresponding to the SRB from ASPI. The CCB contains all important information for performing the instruction, and keeps a buffer ready for the returned data (depending on the function). You will find the specific CAM instructions in Table 31.19.

Code	Function
00h	No operation (NOP)
01h	Perform SCSI I/O function
02h	Determine unit type
03h	Scan SCSI path
04h	Release SIM queue
05h	Set asynchronous reply
06h	Set unit type
07h–Ofh	Resewed
10h	Interrupt SCSI command
11h	Reset SCSI bus
12h	Reset SCSI unit
13h	Interrupt I/O operation
14h–lfh	Resewed
20h	Sense unit
21h	Perform unit request
22h–2fh	Reserved
30h	Activate LUN
31h	Perform target I/O
32h–7fh	Resewed
80h–ffh	Manufacturer dependent

Table 3119: ***The*** CAM instructions

Contrary to ASPI, a CAM function is not activated by a far call to the jump point of the driver, but by calling up INT 4fh, function 8100h. You can see the format of the call in Table 31.20.

Using the function 8200h of INT 4fh, you can determine whether or not a CAM driver is installed. Table 31.21 shows the format of the call.

Register	Call value	Return value
AH		Error code*'
AX	81001	
BX	CCB-Offset	
ES	CCB-Segment	

*) 00h=o k , 01h=invalid CCB Address

Table 31.20: *Call format of INT 4fh, function 8100h – transfer* CCB *to CAM driver*

Register	Call value	Return value
AH		Error code"
AX	8200h	
CX	87651	9abch[2]
DX	cba9h	5678h[2]
ES:DI		Far pointer to "SCSI_CAM"[2]

[1] 00h=CAM driver installed
[2] Only if CAM driver is installed (AH=00h)

Table 31.21: *Call format of INT 4fh, function 8200h – installation* check *for a CAM driver*

TARGA

TARGA is a **manufacturer-independent** SCSI unit driver which, like INT 13h, is constructed as an interrupt interface. This means that all function calls using interrupt INT 78h will be performed, and all parameters in the processor registers will be transferred. Thus, TARGA has no data structure in memory, unlike CAM and ASPI. All of the TARGA functions are listed in Table 31.22.

Code in **AH**	Function
00h	Set I/O port
01h	Determine I/O port
02h	Set DMA channel
03h	Determine DMA channel
04h	Set SCSI unit number
05h	Determine SCSI unit number
06h	Set/reset early return mode
08h	Adapter self-test
09h	Reset SCSI bus
10h	Send SCSI command
11h	Send SCSI command and receive data (PIO)
12h	Send SCSI command and transfer data (PIO)
13h	Send SCSI command and receive data (DMA)
14h	Send SCSI command and transfer data (DMA)
15h	Finish data transfer (DMA)

Table 31.22: *Functions of* TARGA *interrupt* 78h

31.7.5 Different SCSI Standards

Finally, I would like to bring together the attributes and expansions of each of the different SCSI standards.

SCSI-I

SCSI as a standard was born with the (subsequently named) SCSI-I specification. It was originally only an interface for disk drives. Many things remained somewhat vague, and so compatibility problems were still the order of the day. However, SCSI-I did implement a synchronous transfer rate as an option. As the instructions from each equipment manufacturer were at first given too much freedom for the actual implementation (far too much in fact), a standardization was achieved with the Common Command Set (*CCS*). CCS is an extension to SCSI-I, and is not an actual SCSI standard itself. SCSI-I contains only an 8-bit SCSI bus; the transfer rate in asynchronous mode is a maximum of approximately 3 Mbytes/s, or in optional synchronous mode 5 Mbytes/s.

SCSI-II

SCSI-II represents a great step towards an equipment-independent and, therefore, very flexible bus interface. Not only are the instructions and parameter lists clearly defined, and the synchronous mode laid down as compulsory, but in addition, ten classes of equipment with their corresponding required and optional instructions have been defined. In this way, such different equipment as hard disks, CD-ROMs, magneto-optical disk drives, scanners, streamers, media-changers (the good old Juke Box), printers, communication equipment and general SCSI processor systems like a host adapter can be connected and controlled. Thus, SCSI-II is the first «true» SCSI (if you consider the original intention). The two subsequent standards are also embedded into SCSI-II.

Fast SCSI

With Fast SCSI, the maximum clock frequency in synchronous mode is increased to 10 MHz: with an 8-bit SCSI bus, this corresponds to a data transfer rate of 10 Mbyte/s. Note, however, that the SCSI instructions and also the messages, as previously, are transferred asynchronously.

Wide SCSI

In the age of 32- and 64-bit EISA/MCA and local bus systems, the eight bits of the SCSI data bus appear quite modest. Not surprisingly, the width of the SCSI bus has also been increased to 16 or even 32 bits. The relatively long SCSI cables set the frequency limits, not the quantity of parallel lines (ignoring the additional electronic complexity). To be able to use the additional data bits as well, a further so-called *B-cable* is necessary.

Wide and normal SCSI devices can be operated on the same SCSI bus. During a message phase (asynchronous 8 bit), the initiator and the target negotiate as to whether an asynchronous or synchronous, normal or fast mode, 8-, 16- or 32-bit transfer should be performed. A faster or slower mode is selected depending on the performance capabilities and the actual implementation of the applicable SCSI units. Fast and wide modes can be simultaneously active; the

maximum transfer rate at 10 MHz and with a 32-bit width then increases to 40 Mbytes/s. This is a very respectable rate for an external bus.

Ultra SCSI

The more and more reliably operating electronic devices also allow for external buses to increase the clock frequency. This is the background of Ulha SCSI. The transfer clock is doubled to 20 MHz. Unfortunately, the cables must not be as long as for ordinary (Fast) SCSI. In general, doubling the frequency means halving the cable length; if your brand new Ultra drive doesn't want to operate in 20 MHz mode, the reason may be that the cable is too long.

SCSI-III

Currently, the follow-on specification for the very successful SCSI-II standard is in the planning stage. Protocols for the supporting of serial interfaces, multi-host systems and also fibre optic lines and graphics units are planned. In addition to an increase in the data transfer rate (fibre optics), new classes of equipment are also expected.

31.8 Optical Mass Storage

For several years (and especially since the appearance of the CD in the audio and video fields) a triumphant progress has been predicted for optical mass storage. But as with all great prophesies – the reality is usually far more leisurely (fortunately, if we consider the daily 'end-of-the-world' prophesies!). The fact that optical mass storage hasn't already superseded magnetic media, and especially hard disks, is largely due to two reasons: the storage capacity of the hard disks has increased remarkably in the past five years, and further, rewriteable and high-capacity optical data carriers have been under development for years, or very expensive.

Today most optical drives have a SCSI. Only a short time ago, manufacturers often used their own interface so that in addition to the drive, an adapter for the bus slot also had to be provided. You may access the data on the optical mass storage, in principle, in the same way as that on a SCSI hard disk. Thus, there are three possibilities in total:

- INT 21h file and directory function: for that purpose, most drives or controllers/host adapters come with a driver (for CD-ROMs it is usually called MSCDEX.SYS – Microsoft's CD-ROM driver).
- ASPI interface: through the ASPI functions you can issue SCSI commands to access the volume in the optical drive.
- Programming the host adapter directly: this is only possible if you know the adapter very well; for example, you know the memory or 1/0 addresses of the control, status and data registers.

Because programming an optical drive with a SCSI interface is (except for the different SCSI device class and the assigned commands) not very different from programming a hard disk drive, this is not detailed here. If you use INT 21h, the differences between CD-ROMs, hard